AD-A009 939

# SYNTAX, SEMANTICS, AND SPEECH

William M. Woods

Bolt Beranek and Newman, Incorporated

Prepared for:

Advanced Research Projects Agency

April 1975

Unclassified
Security Classification

AD-A 009 939

**DOCUMENT CONTROL DATA - R & D**

*(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)*

| | |
|---|---|
| 1. ORIGINATING ACTIVITY (Corporate author)<br>Bolt Beranek and Newman Inc.<br>50 Moulton Street<br>Cambridge, MA 02138 | 2a. REPORT SECURITY CLASSIFICATION<br>none<br>2b. GROUP |

3. REPORT TITLE
SYNTAX, SEMANTICS, AND SPEECH

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Technical Report

5. AUTHOR(S) (First name, middle initial, last name)

William A. Woods

| 6. REPORT DATE | 7a. TOTAL NO. OF PAGES | 7b. NO. OF REFS |
|---|---|---|
| April 1975 | 57 | 42 |

| | |
|---|---|
| 8a. CONTRACT OR GRANT NO.<br>N00014-75-C-0533<br>b. PROJECT NO.<br>c. Order No. 2904<br>d. Program Code No. 5D30 | 9a. ORIGINATOR'S REPORT NUMBER(S)<br>BBN Report No. 3067<br>A.I. Number 27<br>9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) |

10. DISTRIBUTION STATEMENT
Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce for sale to the general public.

| | |
|---|---|
| 11. SUPPLEMENTARY NOTES<br>Reproduced by<br>NATIONAL TECHNICAL INFORMATION SERVICE<br>US Department of Commerce<br>Springfield, VA. 22151 | 12. SPONSORING MILITARY ACTIVITY<br>ONR<br>Department of the Navy<br>Arlington, Virginia 22217 |

13. ABSTRACT

Recently, speech understanding research has taken a direction which recognizes the importance of syntactic and semantic constraints as an essential part of the process which deciphers speech signals into sequences of sounds (see Newell et al. 1973). Consequently, it has become important for speech researchers to be acquainted with the work that has been done in the area of computational linguistics, attempting to construct computer programs to model the process of natural language understanding. This paper attempts to provide an introduction to the techniques and results which have come out of work in computational linguistics which have special relevance to the design of speech understanding systems. The paper was written for an audience with some understanding of the nature of speech signals and the difficulties of performing an acoustic and phonetic analysis of such signals but with little familiarity with the techniques for parsing and semantic interpretation of natural language or the ways in which such techniques could be used in a total speech understanding system. However, readers with interests in computational linguistics, linguistics, and artificial intelligence may also find the paper of interest.

This paper is not intended to be a survey. Rather, in it I will try to trace the development of what I think are several important ideas and trends in parsing and syntax and in semantic interpretation. I will attempt to convey a feeling for what I think the state of the art is, how it developed conceptually, and some of the new perspectives that the problems of speech understanding place on the processes of parsing and semantic interpretation.

DD FORM 1 NOV 65 1473 REPLACES DD FORM 1473, 1 JAN 64, WHICH IS OBSOLETE FOR ARMY USE.



Unclassified
Security Classification

Unclassified
Security Classification

| 14. KEY WORDS | LINK A | | LINK B | | LINK C | |
|---|---|---|---|---|---|---|
| | ROLE | WT | ROLE | WT | ROLE | WT |
| Grammars | | | | | | |
| Parsing | | | | | | |
| Parsing Algorithms | | | | | | |
| Semantic Interpretation | | | | | | |
| Semantic Networks | | | | | | |
| Semantics | | | | | | |
| Speech | | | | | | |
| Speech Recognition | | | | | | |
| Speech Understanding | | | | | | |
| Syntax | | | | | | |



Security Classification

BBN Report No. 3067
A.I. Report No. 27

# SYNTAX, SEMANTICS, AND SPEECH*

William A. Woods

April 1975

Sponsored by
Advanced Research Projects Agency
ARPA Order No. 2904

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-75-C-0533.



*To appear in: Speech Recognition: invited papers presented at the IEEE Symposium, D.R. Reddy (ed.), Academic Press (1975).

ib

# INTRODUCTION

Recently, speech understanding research has taken a direction which recognizes the importance of syntactic and semantic constraints as an essential part of the process which deciphers speech signals into sequences of sounds (see Newell et al. 1973). Consequently, it has become important for speech researchers to be acquainted with the work that has been done in the area of computational linguistics, attempting to construct computer programs to model the process of natural language understanding. This paper will attempt to provide an introduction to the techniques and results which have come out of work in computational linguistics which I think have special relevance to the design of speech understanding systems. The paper was written for an audience with some understanding of the nature of speech signals and the difficulties of performing an acoustic and phonetic analysis of such signals but with little familiarity with the techniques for parsing and semantic interpretation of natural language or the ways in which such techniques could be used in a total speech understanding system. However, readers with interests in computational linguistics, linguistics, and artificial intelligence may also find things of interest herein. For the reader with little or no background in the nature of speech production and the characteristics of speech signals, I suggest the papers by Denes and Pinson (1963) and by Jakobson, Fant and Halle (1967) as appropriate introductions. This paper should be readable however without such prior knowledge of speech characteristics.

This paper is not intended to be a survey. Rather, in it I will try to trace the development of what I think are several important ideas and trends in parsing and syntax and in semantic interpretation. I will attempt to convey a feeling for what I think the state of the art is, how it developed conceptually, and some of the new perspectives that the problems of speech understanding place on the processes of parsing and semantic interpretation.

## Part 1. Syntactic Analysis

There are two parts to the problem of syntactic analysis -- one is a component of judgment or decision (whether a given string of words is a sentence or not) and the other is a component of representation or interpretation (deciding what the pieces of the sentence are and how they relate to each other). In speech understanding we will see that both of these are important.

Let me start with a mini-history describing what I think the current state of the art is, how it developed conceptually, and some of the new perspectives that the problems of speech understanding place on the evaluation of parsing techniques.

### Phrase Structure Grammars

The field of linguistics was given a great stimulus when the two aspects of syntax (judgmental and structural) were combined in the formalism of phrase structure grammar. Prior to this development, largely due to Chomsky (e.g. Chomsky, 1965), the mechanism whereby a computer program could decide whether a given sequence of words was a grammatical sentence or not would have been difficult to imagine.

The principal component of a phrase structure grammar is a collection of "rewrite rules" such as the following:

```
S -> NP VP
NP -> DET N
VP -> V NP
```

Intuitively, the first rule indicates that a sentence can consist of a noun phrase followed by a verb phrase. Formally, it indicates that in the course of deriving a sentence, one can replace an occurrence of the symbol S in the string derived so far, with the sequence of two symbols NP VP. Similarly, one can replace the NP with the sequence DET N and the VP with the sequence V NP, ultimately deriving the sequence DET N V DET N, which is the sequence of syntactic word categories underlying a sentence such as

The man bit the dog.

## Parsers and Recognizers

The rewrite rules of a phrase structure grammar can be used to characterize the set of possible sequences of words which can be considered grammatical sentences, thereby formally representing the judgmental part of syntax. A formal algorithm for taking a grammar and deciding whether a sequence of words is a sentence with respect to that grammar is called an acceptor or a recognizer.

If in the course of deriving a sentence according to the rules we keep track of which symbols were rewritten into which sequences, one can construct a tree structure such as that represented in figure 1 which gives a very nice representation of what the parts of the sentence are and how they are put together, thus achieving a structural representation of the sentence. An algorithm for constructing such a representation while accepting or recognizing a sentence is called a parser.

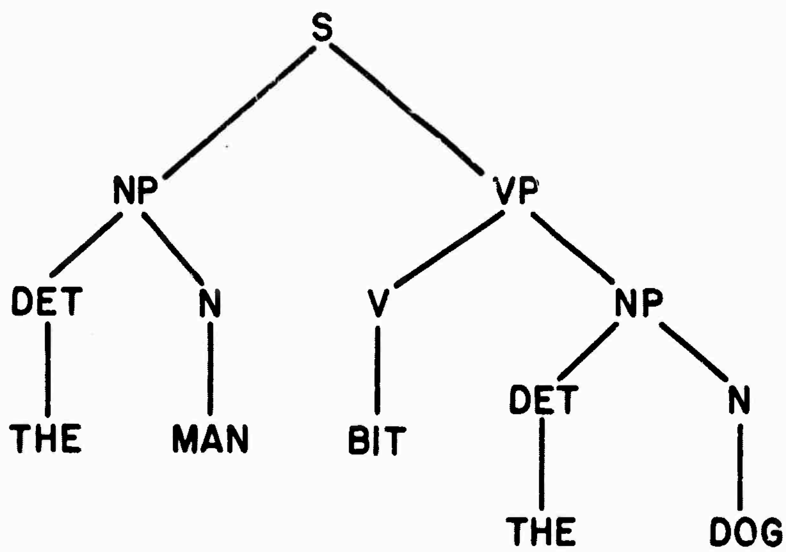



Figure 1: A Sample Phrase Structure Tree

## Lexical categories and dictionaries

Notice that in figure 1 and in the grammar rules there are two different kinds of names of nodes; there are "nonterminal" symbols like S, NP, and VP, which name whole phrase types, and there are other symbols which are essentially lexical word class names, like determiner, noun, and verb. This distinction between terminal and nonterminal symbols is formalized by dividing the vocabulary of special

symbols of a phrase structure grammar into a terminal and nonterminal vocabulary. The initial symbol S, and all of the symbols which later get rewritten by phrase structure rules are in the nonterminal vocabulary. The derivation of a sentence stops when the string consists entirely of terminal symbols. In a simple view of phrase structure grammars, the terminal symbols would be the English words themselves, but this would result in a huge set of "singleton" rules such as:

DET -> the

(on the average there would be several such rules for each word in English). Instead, the syntactic word classes usually serve as the terminal vocabulary and the correspondence between syntactic word classes and the words themselves is taken care of by a dictionary.

## Other Grammar Models

All of the above presentation has been a description of what is called context free phrase structure grammars. There are in fact many different types of phrase structure grammars depending on the types of rules permitted and the way that they are applied. For each different type of grammar, there is a corresponding class of languages which can be characterized by grammars of that type; the grammar formalism is said to generate this class of languages. Whenever two formalisms, either grammars or automata, generate the same class of languages, they are said to be generatively equivalent or equivalent in generative power, and if one formalism generates a superset of the class generated by another formalism, then that model is said to be stronger in generative power. There is a well-known hierarchy of successively more powerful phrase structure grammar models, known among formal language theorists as the Chomsky hierarchy. I would like to introduce these here because I want to come back occasionally and refer to the various things which the different models can do.

The grammar models in the Chomsky hierarchy are known as type 0, type 1, type 2, and type 3 grammars. The context free grammar which we have just described is the type 2 grammar and is characterized by the fact that the left-hand sides of its rewrite rules consist of a single nonterminal symbol and the right-hand sides may be any nonempty string of terminal and nonterminal symbols. The type 3 grammars, also known as finite state grammars, are more restricted than the context free grammars and correspond in generative power to finite state machines. They are characterized by rewrite rules whose left-hand sides are single nonterminals

and whose right-hand sides are either a single terminal symbol or a terminal symbol followed by a single nonterminal.

At the other end of the spectrum are the type 0 grammars, also known as general rewriting systems, which correspond in generative power to Turing machines. General rewriting systems are characterized by rewrite rules whose left-hand and right-hand sides can be arbitrary strings of terminal and nonterminal symbols subject only to the constraint that terminal symbols cannot be rewritten as some different terminal or nonterminal symbol. Type 1 grammars, also known as context sensitive grammars, are strictly less powerful than general rewriting systems and strictly more powerful than context free grammars. They are characterized by rewrite rules in which the left-hand sides specify not only a nonterminal symbol to be rewritten, but also a context of terminal and nonterminal symbols which must be present in order for the rule to be applied.

Figure 2 gives a summary of the types of rules for each class of grammars.

In the figure, the notation V is used to represent the union of the terminal and nonterminal vocabularies of the grammar (Vt and Vn), and the * operator is used to indicate the set of all possible strings which can be made from a given vocabulary (i.e. Vt* indicates the set of all possible terminal strings). The symbol e represents the empty string (i.e. the string with no symbols).

TYPE 0: GENERAL REWRITING SYSTEM

$$\alpha \rightarrow \beta \qquad \alpha, \beta \in V^*$$

TYPE 1: CONTEXT SENSITIVE

$$X \rightarrow \gamma / \alpha \text{ — } \beta \qquad X \in V_N$$

OR

$$\alpha X \beta \longrightarrow \alpha \gamma \beta \qquad \alpha, \beta, \gamma \in V^* \quad \gamma \neq e$$

TYPE 2: CONTEXT FREE

$$X \rightarrow \gamma \qquad X \in V_N, \gamma \in V^* - \{e\}$$

TYPE 3: FINITE STATE

$$X \longrightarrow a\ Y \qquad X, Y \in V_N$$

$$X \longrightarrow a \qquad a \in V_T$$

Figure 2: Summary of the Chomsky Hierarchy of Phrase Structure Grammars

Each of the grammars in the Chomsky hierarchy represents a restriction in generative power (with an attendant ease in parsing or recognition) over the power of grammars with a lower number. Each class with a higher number represents a special case of the classes with lower numbers. The principal difference between the context sensitive grammar and the general rewriting system is that the former is prohibited by the nature of its rules from erasing anything from the working string as it proceeds (i.e. the right-hand sides of rules are always at least as long as the left-hand sides). For the general rewriting systems, this is not the case, and arbitrary amounts of intermediate "scratch work" can be erased out of a

derivation without leaving a trace in the resulting string that is generated. This is what gives the general rewriting system its power, and also has the undesirable consequence that a recognition or parsing algorithm cannot be guaranteed to exist for general rewriting systems. For all of the other classes of grammars, it is possible to construct a recognizer which for an arbitrary string will say yes-or-no whether that string is in a given grammar. General rewriting systems are therefore not very desirable as machine models of language due to this inability to guarantee a recognition algorithm.

## Derivations

For each of the type 1, 2, and 3 grammars, formal parsing algorithms can be devised which, given a grammar and a string, can answer the question whether the string is a sentence with respect to the grammar. This is done by attempting to discover a derivation of the string from the initial symbol of the grammar by means of the rewrite rules. A derivation is essentially a sequence of working strings starting with the initial symbol, each of which results from the preceding one by one application of a rewrite rule. A string is said to be generated by the grammar if there is a derivation of the grammar leading to it.

Figure 3 gives a sample derivation of the sentence in Figure 1.

SUMMARY OF DERIVATION

$$S \xrightarrow{*} \text{DET N V DET N}$$

INTERMEDIATE STRINGS

S

NP VP

DET N VP

DET N V NP

DET N V DET N

Figure 3: A Sample Derivation

Notice however that there can be several distinct derivations for a single phrase structure tree corresponding to different orders of applying the rewrite rules. For example, if instead of expanding the subject noun phrase before the verb phrase one were to expand the verb phrase first, one of the derivations of Figure 4 would result. (Figure 4 compactly represents all of the possible derivations of this particular surface string, with the common initial parts of different derivations combined. Alternative choices for expanding a given string are indicated by the arrows, and individual derivations are terminated by underlining.)

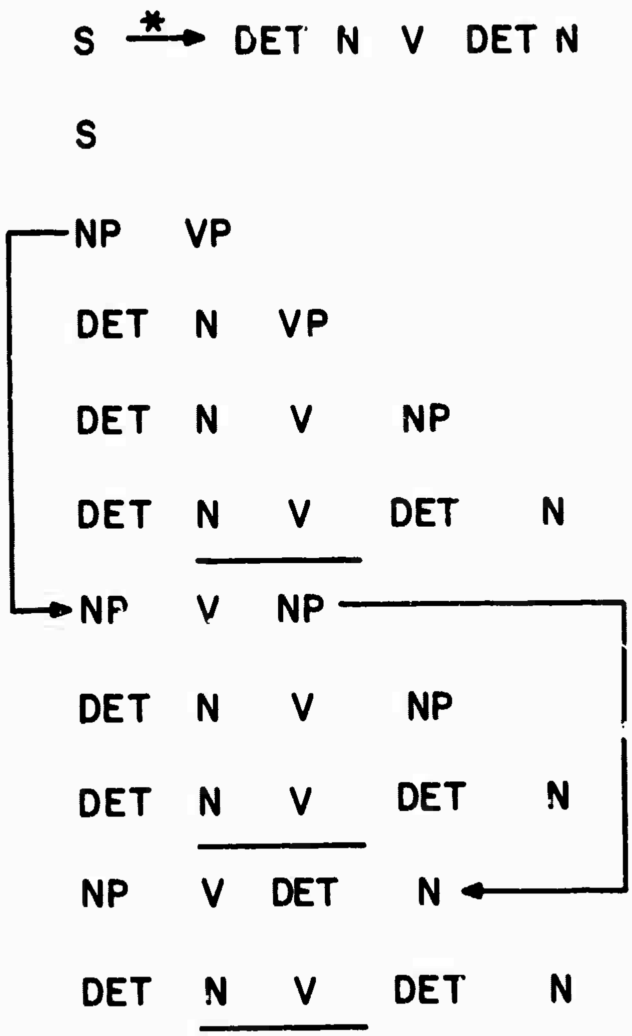



Figure 4: Alternative Derivations of the Sentence from Figure 3

Essentially all of the expansion that appears in the phrase structure tree could be done in any order and each different ordering would give a different derivation which corresponds to effectively the same parse. If we don't want to be swamped with alternative derivations of the same parse, then we need to include in our parsing algorithm some control strategy that will keep it from getting all of them. The typical control strategy that is used in text-based parsers (as opposed to speech) is to decide arbitrarily that the

only derivations which will be considered will be those which expand at each step the leftmost nonterminal in the string. This effectively selects one canonical derivation for each possible parse tree. This makes the derivation shown in Figure 3 the canonical one, and the other two that are shown in Figure 4 are not found.

## The Roots of Nondeterminism

The control strategy which we have just described is very simple to state in terms of a generative rule, but if one wants to use it for an analysis algorithm, it seems to suggest the following analysis strategy: as you start scanning along the string, as soon as you find a piece that matches the right-hand side of some rule, then you can collapse that into a single constituent. However, this strategy will not work in general, as we can illustrate with the grammar of Figure 5. This figure illustrates a very simple grammar for arithmetical expressions. In it, an expression (E) can be a term (T) plus a term or can be just a single term. Likewise a term can be a factor (F) times a factor or just a single factor, and factors can be any of the symbols a, b, or c. Figure 6 shows the structure that we would like to get as a parsing of the string "a+b*c".

$$E \longrightarrow T + T$$

$$E \longrightarrow T$$

$$T \longrightarrow F * F$$

$$T \longrightarrow F$$

$$F \longrightarrow A, B, C$$

Figure 5: A Simple Grammar for Arithmetic Expressions

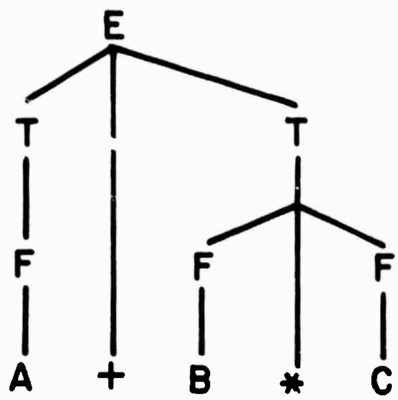



Figure 6: A Parse Tree for "a+b*c"

The way that we have written the rules of the grammar forces on us the priority that the product comes first and then the sum. (A slightly more expanded grammar would include parentheses to enable one to express the other interpretation if that was what was intended.) Now suppose we took this string of characters and the context free rules of Figure 5 and started doing reductions on the string wherever we could. We could reduce the a to an F and then to a T, then we'd have to go on to the + which can't reduce by itself. We could reduce the b to an F and then to a T and then we could reduce the T + T to a single E. After that we would reduce the c to an F and then to a T and after that we would be stuck because there is no rule which will reduce E * T to anything. The structure that we have built when we come to the impasse is shown in Figure 7.

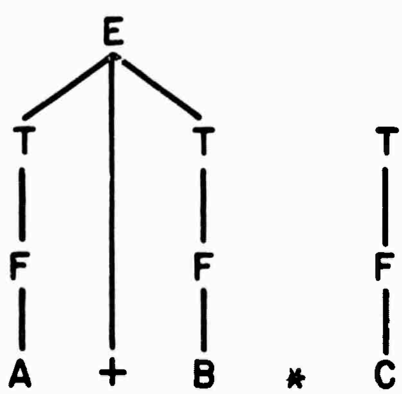



Figure 7: A Blocked Reverse Derivation from "a+b*c"

Essentially, in order to obtain the parse tree in Figure 6, it is necessary not to go ahead and reduce the second F to a T. Instead we must postpone that until reducing the c to an F and reducing the F * F to a single F, which can then be reduced to a T and the T + T reduced to a single E.

So one finds that when one is attempting to do an analysis there is a fundamental nondeterminism that must be provided for. One comes to a place where a rule could be applied, but doesn't know whether that's the right place to make the reduction or not. It is necessary to consider both alternatives, follow out the rest of the analysis, and see which one (if any) of the alternative sequences of choices will give a complete parse tree. This is the first of several sources of nondeterminism in both text and speech parsing.

## Nondeterministic Algorithms

There are many applications in computer science, especially in artificial intelligence and language processing, where systematic search in a space of alternative possible choice is required. A conceptual device for devising algorithms for such tasks is the notion of a nondeterministic algorithm or nondeterministic machine. By this, we do not refer to an algorithm whose behavior is unpredictable, but rather to an abstract algorithm in which there is a primitive choice operation which can make one of several choices. This algorithm is then simulated on a real machine by systematically considering all possible sequences of alternative choices of the abstract nondeterministic algorithm. The nondeterministic machine is a conceptual device to enable the writer of a grammar or other such search algorithm to think of the machine as if it were magically making the right choices, freeing him from explicitly keeping track of the alternative choices and cycling through them. One says that a string is accepted by a nondeterministic algorithm if any of the alternative computation paths lead to a successful analysis.

The first fundamental idea that I would like you to remember is this notion of a nondeterministic algorithm as a device for coping with this type of search in a space of alternative possibilities.

Backtracking vs Parallel Search

There are two principal ways of writing simulators for nondeterministic programs -- one is called backtracking and its effect is that whenever the program is about to make a choice, it saves somewhere (usually on a pushdown stack) all of the information that is about to be destroyed by the choice so that the simulator can come back later, undo it, and try another choice. The program then parses like a deterministic parser until it encounters a blocked configuration such as the one in Figure 7, at which point it undoes the last choice made and tries the next possible alternative. If there is no other alternate choice, then it undoes the next to last choice, and so on until all possible choice sequences have been considered. Floyd (1967) gives an efficient general technique for implementing backtracking simulators for nondeterministic algorithms. In the case of Figure 7, the result of backtracking would be to undo the last reduction of F to T. Finding nothing else to do, the parser then would undo the reduction of c to F, then undo the reduction of T + T to E, and eventually would back up to the point where the b had been reduced to F, but that F had not been reduced to T. The parser could then go on to reduce the c to an F (a second time -- this was done before on the blocked path) and then reduce the F * F to T, which puts us on the right path for the correct analysis.

A backtracking algorithm does its search by systematically working on one path of the nondeterministic algorithm, saving enough to undo it later. The systematic way in which it walks through the space of possible choices is called "depth first". That is, after making one choice, it proceeds to make a choice that depends on that one, and another that depends on that, and so on, building up a stack of other untried alternatives at different "depths". Only when it encounters a blocked configuration does it undo the most recently made choice, and it tries all possible choices at that "depth" before backing up to the next previous level on the stack of alternatives. If the space of alternative choice sequences were laid out as a tree, the backtracking search would correspond to a left-first tree walk.

Another way of handling nondeterminism is by what I'll call independent alternatives. In such a program, every time that you are about to make a choice, you create an object for each of the possible choices. This object corresponds to a state or configuration of the hypothetical nondeterministic machine which you are simulating. In a real machine, a configuration is basically the contents of the program counter and the register contents; in the simulation of a nondeterministic machine there are many such configurations instead of just one. (This is similar to what goes on in a time sharing system.) For a

nondeterministic finite state machine, the configuration is basically the state that you are in and the point in the input string that you have gotten to. In programming a system for handling independent alternatives, every time that you come to a choice point, you make up as many configurations as there are alternative choices, and you are now free to work on those configurations all in parallel (or "breadth first") or you can jump around from one to another (working on the ones that seem most likely of success before working on others for example). With multiple independent alternatives, you can pick up a configuration, determine its state, look at where it is in the input, compute the next configurations which it could get to, and then go to another configuration (which may or may not be one of the ones you just created). Just like a time-sharing system you can run a lot of these configurations in pseudo parallel with varying priorities for service.

There is a tremendous advantage for speech understanding and also in text parsing for implementing nondeterministic programs in terms of independent alternatives rather than backtracking. With independent alternatives, if you are in a position where it is difficult to decide which of the alternative choices is the best to follow, it is possible for you to follow several parsings in parallel, or to jump from one to another depending on which looks better at any given moment. In the backtracking approach, one has to systematically walk down a long path into barren territory before he can walk back to the place where the next best choice is. The only way to go back and consider one of the alternatives to a choice is to plow on ahead to completely search the space on the current path exhaustively and then back up out of it. Once one has left a given path it is not possible to come back to it and push it further. Even in the simple illustration of backtracking for the example in Figure 7, there were two or three things of an unimaginative nature that had to be undone before getting back to where the right alternative choice had to be made. In more complicated examples, the amount of such "wasted" or uninteresting parts of the space that have to be searched before one can get back to the correct place to make an alternative choice can be astronomical.

I will make a pitch then for a second fundamental idea which you should know about -- namely this difference between systematic backtracking and the following of multiple independent alternatives.

## Bottom Up, Top Down, Predictive, and Nonpredictive Parsing

The algorithm that we described above for finding a derivation of a given string by reversing the generative rules of the grammar is an algorithm that is referred to as "bottom-up". That is, we look into the input string or the current working string until we find something that matches the right-hand side of some grammar rule, and then reduce that matching portion by replacing it with the left-hand side of the rule. (I'm assuming a context free grammar here for simplicity.) We apply this process over and over again until we finally reduce the entire string to a single symbol. (At least the goal we are trying to achieve is such a reduction of the string into a single symbol.) Notice that in the statement we have just made, we have not specifically mentioned the systematic consideration of each of the possible rules that could have applied at each step and the different positions in the working string where rules could have been applied. It is exactly this freedom from consideration of detail that is achieved by thinking of the process as a nondeterministic algorithm. Of course the details need to be considered eventually in order to make the algorithm function on a real machine, but these considerations can be made separately and they can be done once and for all for a parsing system and not have to be redone separately for each grammar or version of a grammar which is written.

There is another kind of parsing algorithm at the other extreme which is called "top-down". It gets this name because it starts by expanding the grammar rules "from the top" and only looks for comparison at the input string when a terminal symbol appears in the expansion. A simple version of a top-down parser makes use of a pushdown store into which the initial symbol of the grammar is placed before parsing begins. Subsequently the algorithm proceeds as follows: If the topmost symbol on the stack is a nonterminal, then some rule of the grammar with that nonterminal as its left-hand side is selected (another nondeterministic choice) and the topmost symbol of the pushdown stack is replaced with symbols from the right-hand side of the rule (so the leftmost symbol of the right-hand side is now the topmost symbol of the stack). If the topmost symbol of the stack is a terminal symbol, then it is compared with the next unused symbol of the input string. If they are the same then the topmost symbol of the stack is removed and the string is advanced. If they do not match then this configuration is blocked -- i.e. this path of the nondeterministic search is terminated. The string is accepted if the pushdown stack becomes empty at the same time that the last symbol of the input string is used. (Note again our use of the nondeterministic algorithm to simplify the explanation. In an actual parsing algorithm,

all possible choices of expanding the topmost nonterminal of the stack are pursued and the string is accepted if any of the alternative computation paths leads to the accepting criterion.) An example of a top-down analysis using a pushdown store is shown in Figure 8. (Here the rectangular enclosure represents the pushdown store, the arrows the steps in the analysis, and the plus sign indicates the consumption of a symbol from the input string by a given stack configuration.)

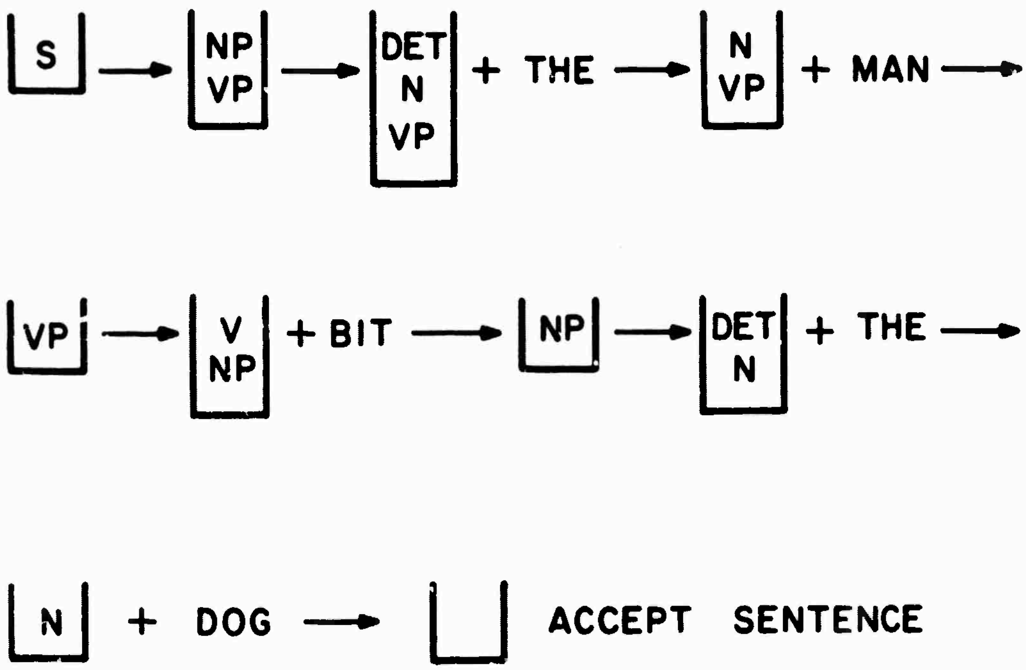



Figure 8: A Sample Top-down Predictive Analysis Using a Pushdown Store

The Harvard Predictive Analyzer

The original Harvard Predictive Analyzer (Kuno and Oettinger, 1963) does a slightly more optimized version of the top-down technique just described. It works with a grammar which has been transformed so that all of its rules have a terminal symbol as the first symbol of their right-hand sides. Thus at every step of the pushdown store analysis the algorithm consumes a symbol from the input string, and the number of steps in a given computation path of the nondeterministic machine is at most n, where n is the length of the input string. (Of course the number of steps

of the real computer in simulating the nondeterministic algorithm is much greater since it has to follow out all possible alternative computation paths.) An additional advantage of the special form of the context free rules used by the predictive analyzer (known as Greibach normal form, or standard form) is that it eliminates the possibility of infinite loops due to the symbol on top of the pushdown stack expanding into a string which eventually results in a new instance of the same symbol on top of the stack without advancing the input string. An algorithm due to Greibach (Greibach, 1967) which converts an arbitrary context free grammar into a standard form grammar finds and eliminates the possibility of such "left-recursion".

## Predictive vs Nonpredictive Parsing

There has been a great deal of discussion in the parsing literature about the differences between top-down and bottom-up algorithms. An example is a paper by Griffiths and Petrick (1965) which characterizes several varieties of each type. However, in recent years there have been a number of parsing algorithms developed which don't fit into either of these broad categories, and I think that the classical distinction between top-down and bottom up is becoming very fuzzy. The distinction which I think is more important -- a distinction which is correlated with the top-down bottom-up distinction for the two simple algorithms presented -- is the distinction between predictive and nonpredictive parsing. A predictive parser is one that will only look at a given point in the input string for things of a sort that it expects to see there, whereas a nonpredictive parser will find a given construction only as a function of the constituents which make it up, irrespective of whether such a constituent is compatible with an analysis of the symbols on either side of it in the input string. For example, an inherent feature of the top-down pushdown store algorithm which I presented above is that at each point in the analysis there exists on the stack a prediction of the types of phrases which are expected to occur to the right of the current point in the input string. As the algorithm operates, only those constituents will be looked for. Contrast this with the situation in the simple bottom-up algorithm. There, if the terminal symbols could be grouped together to form some constituent, then that alternative would be tried regardless of whether there is an analysis of the symbols to the left with which this constituent could combine.

The predictive parsing technique has an advantage for most parsing applications since it considerably reduces the number of applications of rules that have to be considered

and the number of "accidental" constituents that are found (i.e. sequences of words that could make up a constituent in some other context but which are not a constituent of any complete analysis of the current string). For example, in a predictive analysis of "the man bit the dog", using the grammar of Figure 1, the parser looks for a noun phrase at the beginning of the sentence because the grammar says that sentences can begin with noun phrases. However, once it has found the subject noun phrase, it doesn't try to look for a noun phrase at the place that starts with "bit" because there is no grammar rule which would use a noun phrase at that point. In the bottom-up approach, all rules are attempted everywhere since there is no prediction. Not only does this result in more rules that have to be tried, but it also results in more spurious matches that don't lead to correct parsings.

For parsing text in the form of sequences of words, there is a great advantage to using the predictive algorithm because it follows fewer blind alleys. On the other hand, there is a problem in continuous speech understanding which reduces its advantage. In continuous speech understanding, there is a fairly high probability that your guess for the word at any given point in the string may be wrong. This is especially true of the first and last word in the sentence due to phonological effects at the beginning and ends of utterances. If your guess of the first word is wrong, then all of your predictions later will be influenced by it, and if it induces you to only look for those things that will be consistent with that wrong word, then you may never recover the right parse. The nonpredictive parser that goes up and down the string doing everything it can stands a better chance of recovering from such errors. Specifically, it stands a better chance of finding most of the parse in spite of a wrong or missing word. It can then provide this information as a source for prediction as to what the missing word might be or what kind of word is required in a given region.

Another point then that I would like to make is this tradeoff between predictive and nonpredictive parsing algorithms for speech understanding. I don't want to make a strong case that one or the other is better; I want to give a feeling for what the tradeoffs are between the two algorithms. The predictive one will do a more selective search, and if one is confident that the things on which it is basing its predictions are right, then it is preferable. On the other hand, if there's a high chance that they're wrong, then the disadvantage is that the prediction may keep you from finding enough of the correct parse to be a useful source of information for error correction.

### Well-formed Substring Tables

One thing that was found very early in the development of parsing algorithms, especially with the enumerative, top-down, predictive algorithms is that when alternative computation paths are done separately, duplicate work is done on the separate paths. For example, if two possible ways of analyzing the beginning of a sentence cause the analysis to split up into two different computations, the entire remaining analysis will be done twice, even though it may be the same in both cases. A "well-formed substring table" is a mechanism for saving the results of the analysis of a constituent on one path of a nondeterministic computation so that they can be used on other paths without redoing the computation. Whenever in the course of analysis, a complete constituent is found, it is recorded in a table indexed by the type of constituent and the position where it begins. Whenever the algorithm is about to predict a constituent of a given type at a given position, it consults the well-formed substring table to see if such a constituent has already been found, and if so, then the results are used without recomputation.

### Table Oriented Parsing Algorithms

The use of the well-formed substring table is sufficiently useful that some parsing algorithms have been designed exclusively around that notion. Their central purpose is to fill in this table with entries saying there is a constituent of type x from position y to position z in the input. Their acceptance criterion for a string is finding in the table an entry indicating a constituent of type "sentence" from the beginning to the end of the input sequence. In the design of such an algorithm, one looks for a strategy for filling in the table so that whenever, in applying a grammar rule, one needs the answer to a question "Is there an x from y to z", the strategy will already have considered all possible ways of filling that entry in the table, and the answer can be determined by simply examining the content of the [x,y,z] entry of the table. The resulting algorithm consists mainly of walking this matrix in an appropriate order and filling in entries on the basis of other entries and the symbols in the input string. For example an algorithm due to Younger (1966) fills in the entries in order of length of the resulting constituent (and forbids grammar rules whose right-hand sides consist of a single nonterminal). Since the lengths of the constituents which match the right-hand side of a rule will be less than the length of the constituent that will result all the necessary table entries for constituents of a given reduction will already have been made when that reduction is considered. Thus when filling in the table for constituents of length 3 for example, all of the entries for constituents

of length 2 and 1 will already have been made and any questions about the existence of such constituents can be answered by merely consulting the table. The constituents of length 1 are found by matching singleton terminal rules against the input string. When such an algorithm terminates, if there is an entry for the initial symbol from the beginning to the end of the input string, then the string is accepted by the parser, otherwise it is rejected.

Eliminating Redundancy

In the above type of algorithm, it is critical in order not to do a lot of excessive computations that a particular order of filling in the table be used. This is so that one can rely on any answer that is needed having been put there at an earlier point in the sequence. This has many efficiency advantages for ordinary text parsing. However, it has a disadvantage for speech understanding applications, since one of the critical elements early in the chain may be misheard or garbled and thereby keep the rest of the analysis from being found (which could be used to help identify the garbled word). This same disadvantage applies to the left-first canonical derivation of a parse which we mentioned earlier, and to any other parsing technique which requires the individual steps in an analysis to be found in a particular canonical order. If one of the critical things that has to be found first in some such ordering is wrong and if all of the subsequent processing is dependent on it, then it will be very difficult to recover from the error. I think therefore, that it is important for speech understanding to try to relax some of these ordering restrictions. This is a fundamental departure from the way that most text parsing systems operate and it is going to require a different solution to the problem of finding the same parse over and over again in different ways.

In many cases, it may be important to be able to jump over and find the object noun phrase and then the verb phrase when you haven't found the subject yet. For example, in those cases where the subject wasn't findable because of a garbled word, a well understood verb phrase could be used to predict what kind of subject ought to be there. However, in other cases when you have found the subject first on one path, a computation path which finds the verb phrase and then comes back and works on the subject will find the same parsing over again. The solution that we have been using in the BBN system (Woods, 1974) -- the solution which I think has to be used -- is to put in appropriate checks at various choice points to ask whether the thing that is about to be produced has been found already on some other path and avoid creating a duplicate. When this is done at the level of

noun phrases, embedded clauses, etc., it tends to block the redundant generation of larger constituents before the duplication becomes unmanageable. It still carries with it the cost of the additional checking, but I think that this cost is essential in order to cope with the errors that will occur in speech.

## Lexical Ambiguity

I've mentioned a number of things which make the parsing problem for speech understanding more difficult than traditional text parsing. Another difficulty is the ambiguity of word identification in the input sequence of sounds. The major source of lexical ambiguity in text parsing is the possibility of multiple syntactic categories for a given word. In a classical example of sentential ambiguity, "Time flies like an arrow," the word "time" has three possible syntactic categories (noun, verb, or adjective), "flies" can either be a verb or a noun, and "like" can either be a preposition or a verb. If we think of a parser receiving a sequence of these kinds of categories as input, there would be 3x2x2=12 strings of syntactic categories that you could get for this sentence. If you had to put each such sequence through the parser separately (apparently some early parsers did exactly that) you would be doing twelve separate parsings. Imagine what would happen with a sentence of say 20 words with an average ambiguity of 2 categories per word; you would have over 1,000,000 different possible such sequences. In speech understanding, this basic ambiguity is magnified by the inability to unambiguously determine the segmentation of speech sounds into word sequences. Clearly one doesn't want to run a parser on a separate enumeration of each possible sequence of syntactic categories.

## Word Lattices

A technique that has been very effective for dealing with lexical ambiguity has been the use of a lattice of input symbols rather than a single string. A simple example of such a structure is illustrated in Figure 9.

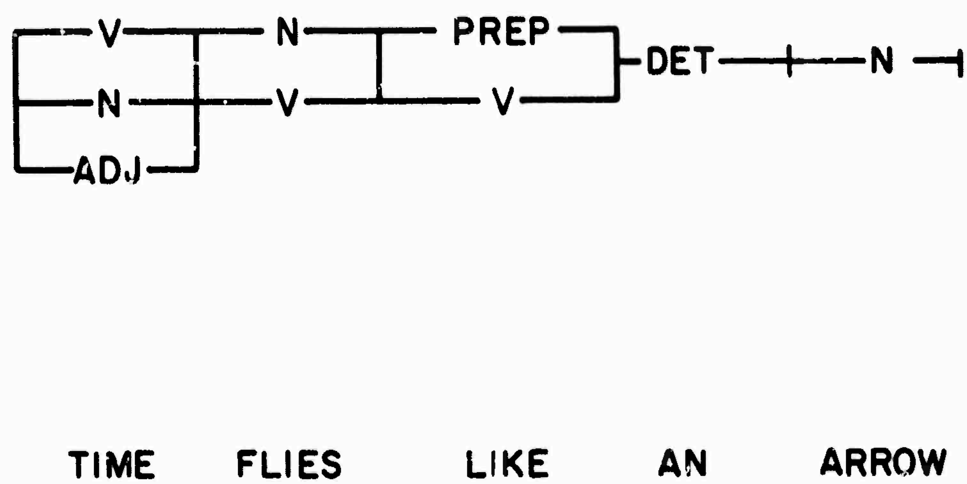



Figure 9: A Sample Word Lattice

Such lattice compactly represents all of the possible alternative sequences of input symbols with the common parts of different sequences factored together so that processing on them needs to be done only once. With such an input, grammar rules are matched the same as before, except that as a rule is matched against the input, particular paths are selected through the word lattice which satisfy the match. This technique has a tremendous benefit in terms of the amount of computation required for parsing. When a particular rule is matched at a given point in the word lattice, all of the possible sequences of words in which the matching sequence occurs are effectively factored together so that the result of the reduction is effectively performed just once for an entire equivalence class of word sequences. This technique is very attractive for speech understanding because the possible alternative segmentations of the input signal into words leads to a lattice structure similar to that illustrated in Figure 9 (although of slightly more varied structure). Whereas the structure in Figure 9 is nothing more than a sequence of alternative syntactic categories, the structures for word lattices in speech understanding tend to have much more branching, and the individual branches leaving a given point do not all come together again at the same point. However, the same parsing algorithm runs on this more generalized input lattice and saves a tremendous amount of processing by avoiding the multiplication of combinatorial possibilities.

## Chart Parsers

The concept of a word lattice for the input symbols and the use of a well-formed substring table for representing the intermediate stages of parsing are closely related, and can be combined into a single data structure in a parsing algorithm. The structure of the well-formed substring table is exactly the same as that of the word lattice, and if it is appropriately indexed by position in the input string (or position in the word lattice) then it shares the property of compactly enumerating for a given position all of the constituents which begin at that position and all of the positions where such constituents end. A classical parsing algorithm known as Cocke's algorithm (see Hays, 1962) and a generalization of that by Martin Kay, which Kay now calls a chart parser (1967), combine a lattice of input segments with a lattice of well-formed substrings in a single data structure called a chart. The goal of such a parser is to expand the initial lattice of input symbols into a complete lattice (or chart) of all of the constituents that can be found in any analysis of any path in the initial lattice. An example of such a lattice for the sentence "Time flies like an arrow" is shown in Figure 10.

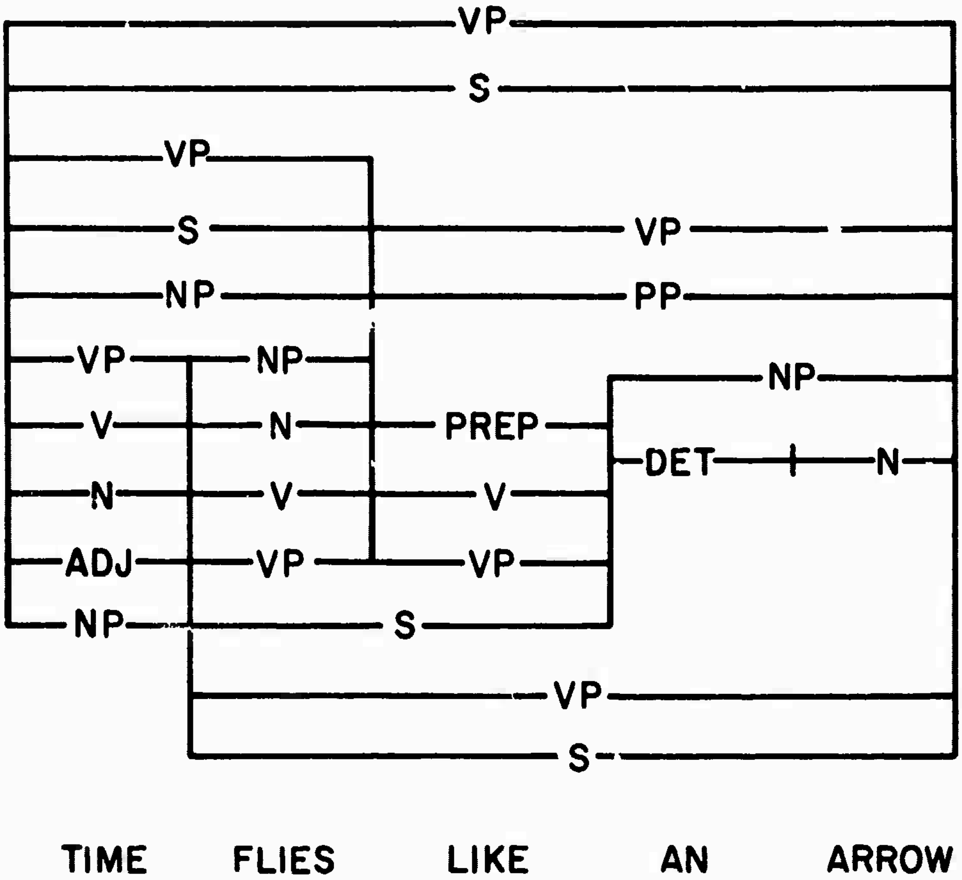



Figure 10: An Example of a Well-formed Substring Lattice or Chart

Each labeled horizontal line in the figure between vertical strikes represents a segment added to the chart as a result of the application of some rule (or one of the initial entries in the word lattice). Both of these parsing algorithms (Kay's and Cocke's) select a particular order for walking the chart and adding new segments as a result of matching rules against the segments already in the chart, and both produce a very nice recognition algorithm that keeps a great deal of the common parts of different analyses merged together. The principal difference between Kay's parser and Cocke's is Kay's generalization of the method to handle general rewriting systems and an approximation to transformational grammars. For strictly context free grammars, both algorithms are effectively the same. In this paper, I will call all such parsers (both Cocke's and Kay's) and their derivatives "chart parsers". In particular, the usual implementation of the classical nonpredictive bottom-up parsing algorithm is a chart parser.

Parsing versus Recognition

In order to be called a parser, an algorithm must not only calculate whether a string is accepted or not, as does a recognizer, but it must also keep a record of the derivation and provide one or more structural analyses of the sentence. In my description of most of the parsing algorithms so far, I have glossed over this distinction and only the recognition aspects have been discussed. In order to be a parser, an algorithm must keep track of and report what constituents were used as pieces of what higher constituents. This can be done conveniently for a chart parser by annotating each of the segments of the chart with a list of the constituents which formed it, -- that is, by a list of the segments which were combined by some rule to produce the annotated segment. In general, there can be several ways to form a given segment from different sequences of constituents so the annotation must provide for several such constituent lists in order to represent all possible analyses.

Both Cocke's algorithm and Kay's are bottom-up, nonpredictive algorithms and share with other such algorithms the property of finding many accidental constituents that do not form a part of any complete analysis. Such accidental constituents clutter up the chart. Figure 11 shows a chart for our example in which all such accidental segments have been removed. Such a "cleaned up" chart together with its constituent pointers provides a very compact representation of all of the possible alternative analyses of the input with the common parts of the different analyses merged together. Figure 12 shows the chart of Figure 11 with constituent pointers added for one particular parsing of the input, and Figure 13 shows the same chart with all constituent pointers included.

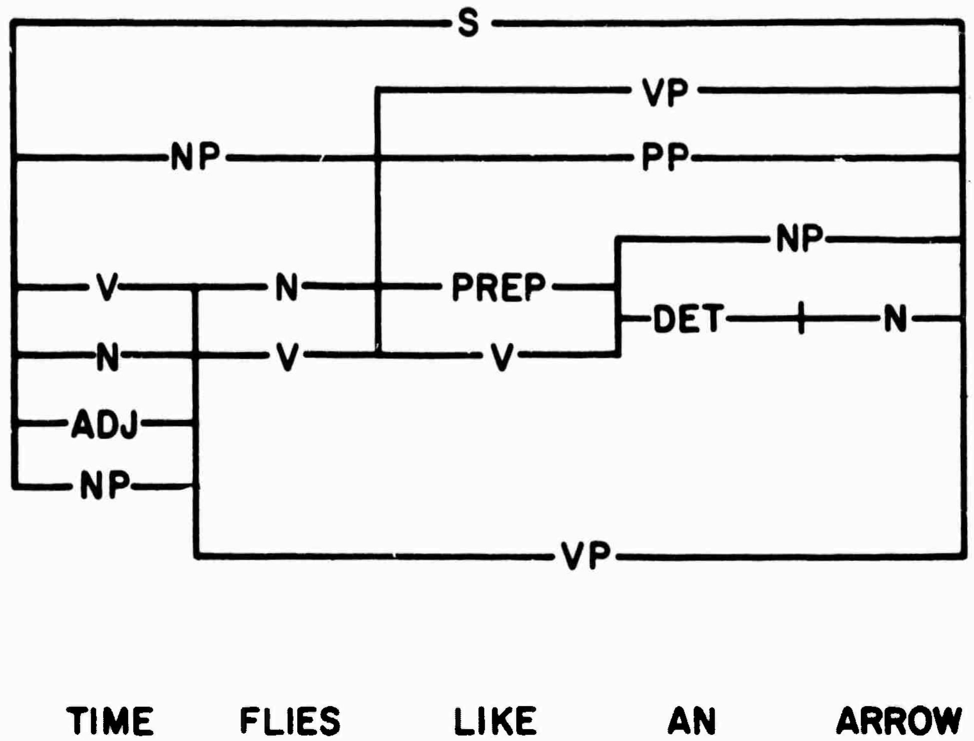



Figure 11: A Chart with Accidental Constituents Removed

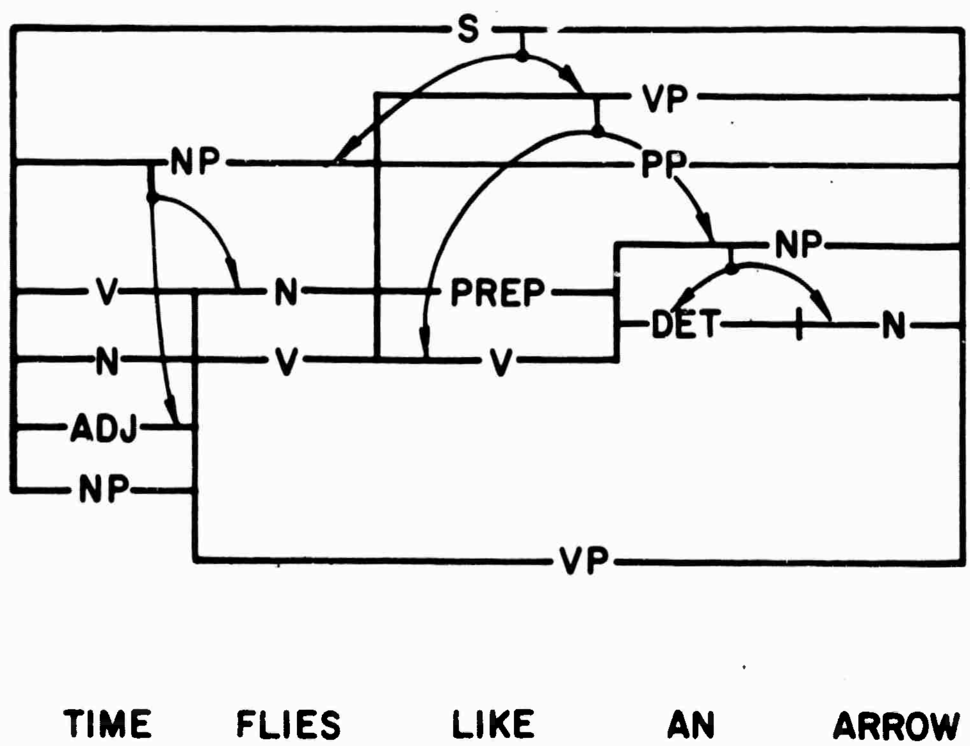



Figure 12: A Chart Showing Constituent Pointers for One Parsing

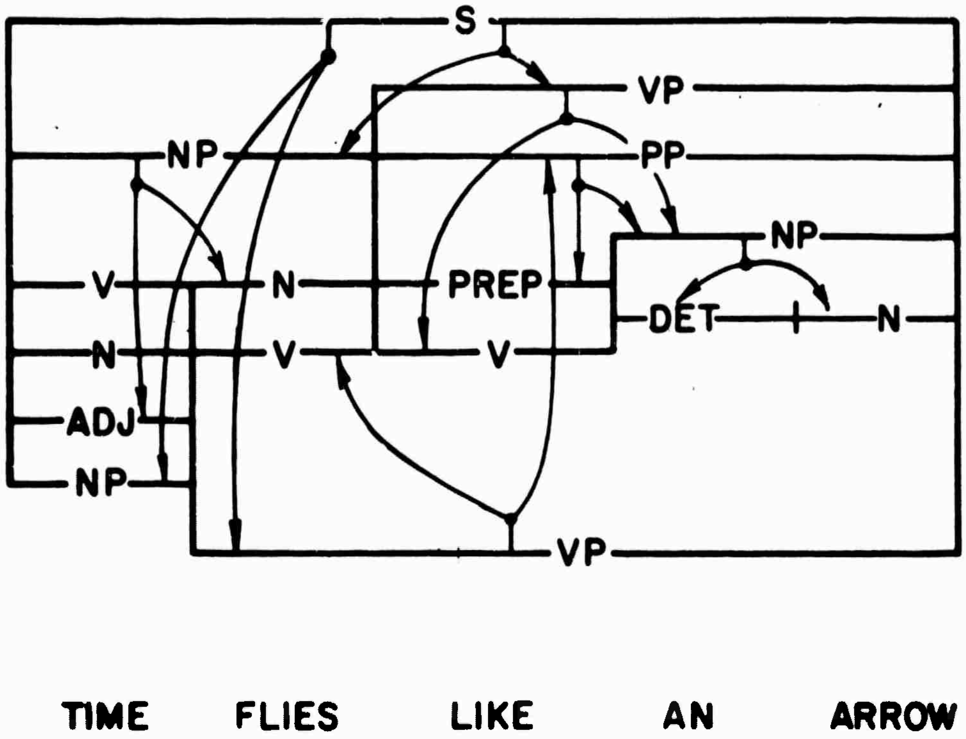



Figure 13: A Chart Showing All Constituent Pointers for Two Parsi

In more typical cases, one cannot draw as nice a picture of the chart as in our example, (in particular it may not be a planar graph), but inside a computer, a table of positions, each with a set of associated segments (indicated by the constituent type and the position where the segment ends) suffices to handle the most general case for a recognizer. For a parser, the inclusion with each segment of a list of alternative constituent lists, each of which is a list of segments (where a segment is named by its left and right end points and the constituent label) suffices to produce a compact representation of all the possible parses.

### Earley's Algorithm

There is another parsing algorithm for context free grammars due to Jay Earley (Earley, 1970), which can be thought of as a predictive chart parser. This algorithm combines the benefits of the systematic, lattice-oriented parsing of the well-formed substring or chart parser with the advantages of predictive analysis. Although the algorithm was developed in the context of parsing for computer programming languages, and is presented as such by Earley, the algorithm has many theoretical advantages for

parsing context free grammars in general, and an appreciation of its operation is important for the understanding of context free parsing. Earley's algorithm does not quite fit into either the top-down or the bottom-up models, or rather it seems to fit equally well into both. Starting from the beginning of the string, it begins to fill in a table (which Earley calls a state table) in which it records for each position in the input string, each rule of the grammar that has been partially matched up to that point or which might possibly match beginning at that point (i.e. each rule that would be consistent with what has been parsed so far to the left of that point). The table is organized into columns, one for each position in the input string, and the procedure for filling out a given column i+1 is as follows:

1. (transition) Make entries in the column for rules that appear in the preceding column and whose match can be continued by matching the input symbol associated with this column.

2. (prediction or "pushing") make beginning entries in this column for every constituent which could be used to continue a match for a rule already in this column (each rule remembers the column in which its match was begun so that when the match is completed, the algorithm can return to the column where the constituent was wanted and continue the match of any and all rules which wanted it. This memory of the column in which a subconstituent match was begun replaces the use of a stack in most predictive algorithms, and gains a combinatorial benefit by not having to enumerate all of the different possible stacks which could sit above a given subconstituent computation. A given constituent in a given place will be looked for and found only once.)

3. (completion or "popping") For each rule whose match has just been completed in this column, go back to the column where that match was begun and pick up and continue the match of all rules which can use the constituent just formed.

In Earley's statement of the algorithm the progress of a rule match is recorded by a pair of numbers -- the rule number and the number of symbols in the right-hand side of the rule which have already been matched. An entry in the table consists of these two numbers plus the number indicating the column in which the rule match was begun. A sentence is accepted if, when the last column is filled out, it contains an entry for a rule whose left-hand side is S, whose match has been completed, and whose match was begun in column 0. (The algorithm begins by initializing column 0 to contain all of the rules whose left-hand sides are S.)

Earley's algorithm is frequently thought of as a top-down parsing algorithm because of the way that it starts with the assumption that it is going to build a sentence and successively elaborates its set of rules to be looked for by passing information "down" in step 2. However, once such top-down prediction (which incidentally may leap over what will amount to many cycles of left recursion in the final analysis) has determined the set of rules to be used at a given point, the subsequent analysis will do almost the same kind of bottom-up structure building as any other chart parser, the principal difference being that for Earley's algorithm we will get a subset of those entries in the chart that would have been produced by an ordinary chart parser. This is because the prediction technique has eliminated all those entries which are not at least consistent with some analysis of the string to the left. Once again, this prediction is a mixed blessing for speech understanding since if the prediction is made on the basis of unreliable evidence, it may keep us from finding enough of an analysis to benefit error correction.

## Transition Network Grammars

The presentation so far has been illustrated by two extremely simple sample grammars. When one begins to write a grammar for any appreciable subset of natural language, one finds that there are some verb phrases which consist of a verb alone, some with a verb plus an object noun phrase, some with a verb, an indirect object and a direct object, any of these three forms with a prepositional phrase added, any of the three forms with two prepositional phrases, etc. If one were to write each of these as a separate context free rule, as illustrated in Figure 14a, we find a very rapid proliferation (possibly infinite) of rules that share a lot of stuff in the right-hand sides. People who write grammars immediately find themselves falling into notations such as that illustrated in Figure 14b in which optional constituents, alternative constituent sequences, and repeatable constituents are indicated by some notation (usually parentheses for optionality, curly brackets or vertical strokes for alternative sequences, and the Kleene star operator (*) for repeatable constituents). These are usually thought of just as abbreviations for a set of ordinary context free rules, but the actual expansion of such notations into ordinary context free rules is a very bad way to implement them. Instead, one buys an advantage in parsing if he takes advantage of these primitive notions of optionality, alternatives, and repeatability. Transition network grammars provide a mechanism for doing this.

A basic transition network (BTN) is essentially a finite state transition diagram to which recursion has been added by fiat (see Woods, 1969, 1970, 1973a). The result is no longer a finite state device, but rather is formally equivalent to a pushdown store automaton or a context free grammar. The BTN is a labeled, directed graph whose nodes, which we call states, represent states which the grammar can be in in the course of generating (or analyzing) a sentence, and whose arcs represent transitions from state to state. The labels on the arcs indicate the input symbol or type of phrase which must be consumed from the input string in order to make the transition. It is the possibility of arcs (called PUSH arcs) labeled with the names of phrase constituents that provides the recursion which makes this model more than finite state. The grammar contains a start state for each of the types of constituents which can be called for on a PUSH arc, and distinguished states called final states which represent the completion of the analysis of some constituent. A PUSH arc can be taken if some string accepted by the start state associated with the label of that push arc is consumed (or generated). There is a mechanical procedure presented in Woods (1969) for transforming any given context free grammar into an equivalent BTN and performing a number of optimizing transformations on the resulting BTN to produce a grammar which is more compact and more efficient for parsing than the original context free grammar. Essentially the BTN provides a way to factor a context free grammar into a finite state part and a recursive part so that as much of the grammar as possible can be expressed in the finite state part and optimized by the same techniques applicable to finite state grammars.

The set of notations used by linguists for representing alternative sequences and repeatable constituents in their grammar rules correspond to the operations called "union", and "closure" in the theory of finite state automata, which together with the operation of concatenation are known to generate the finite state languages. Thus, the right-hand sides of grammar rules using these notations are merely notational variants of what is in automata theory called a "regular expression", and there exist formal procedures for translating such a representation into an equivalent transition diagram for a finite state machine. These same procedures can be used to translate a context free grammar using these notations into an equivalent BTN, such as the one illustrated in Figure 14c.

VP → V

V NP

V NP NP

V PP

V NP PP

V NP NP PP

•

•

•

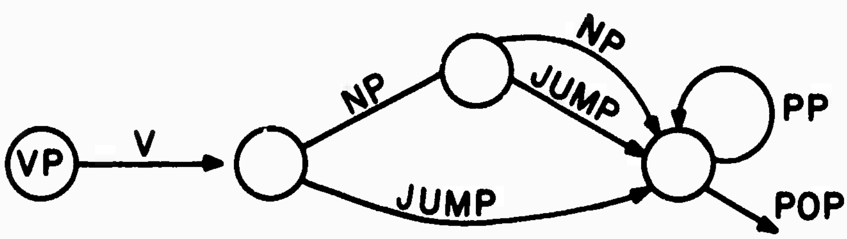
a. SEPARATE CONTEXT FREE GRAMMAR RULES

---

VP → V (NP (NP)) (PP)*

b. MERGED REPRESENTATION

---




c. REPRESENTATION AS BASIC TRANSITION NETWORK (BTN)

Figure 14: Alternative Representations for Multiple Right-hand Sides of Grammar Rules

Thus the BTN formalism provides a realization for these notions of alternative sequences and repeatable constituents that is more efficient for a parser as well as being less redundant as a linguistic specification. Each of the arcs leaving a given state represents an alternative possible continuation of the string being generated (or of the analysis of a given string).

The transition network grammar effectively provides for the merging of common parts of what would be different context free rules, and this permits parsing operations to be performed only once on such parts instead of separately on each individual copy as would be the case if the expressions were expanded into separate ordinary context free rules. Most of the parsing algorithms for context free grammars have natural generalizations to transition network grammars which take advantage of this merging. In particular, Earley's algorithm is a natural algorithm for BTN grammars and the number of parsing operations required by Earley's algorithm for a parsing of an optimized BTN grammar compared to the parsing of an equivalent context free grammar can easily be less by factors of four or five. A presentation of a version of Earley's algorithm for BTN's is given in Woods (1969).

## Grammars for natural English

In comparing the models of the Chomsky hierarchy with each other, it has been found that whereas the finite state grammars have great computational advantages for parsing (there exist formal, mechanical optimizing procedures of various types for finite state machines), the absence of recursion makes it unsuitable for natural language analysis. On the whole, context free grammars provide the simplest and most natural grammars for natural language but are formally incapable of dealing with certain kinds of coordinate constructions and discontinuous constituents. Context sensitive grammars have sufficient formal power to provide a recognizer for such constructions, but provide no useful structural descriptions. General rewriting systems add no useful power not already present in context sensitive grammars and have the undesirable consequence that it is not possible to have a parsing algorithm for the entire class of such grammars.

## Transformational Grammars

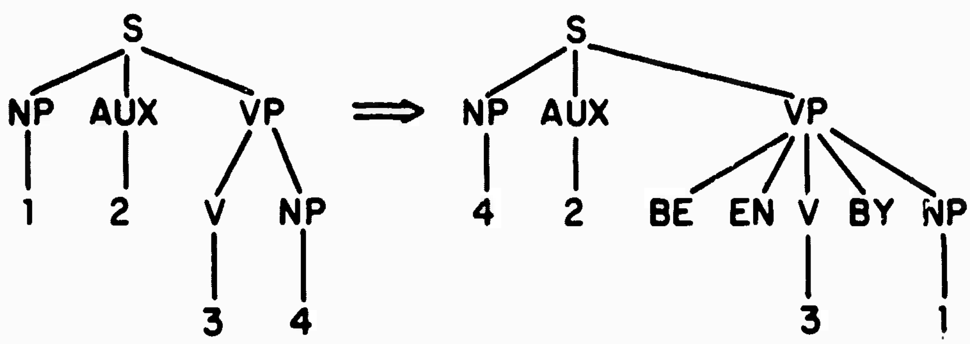
There are a number of other grammar formalisms that have been proposed for natural language which have been shown to be equivalent to the ordinary context free grammar model. One formalism, however, with considerably more power than context free grammars has stimulated linguistics and served as the vehicle for most of the study of natural language grammar in the last decade. This is the transformational grammar of Chomsky. A transformational grammar basically consists of a context free "base" grammar plus a set of transformational rules which can permute the order of constituents and in general move, delete, and insert constituents at various positions in the parse tree. Transformational rules can also test conditions such as identity of constituents and the presence of syntactic features associated with the words and sometimes the phrases of the sentence. Perhaps the simplest example of a transformational rule is the passive transformation shown in Figure 15, which produces the "surface structure" for a passive sentence from the "deep structure" that underlies the corresponding active sentence.

**PASSIVE**

| NP | (AUX) | V | NP | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | ⟹ |
| 4 | 2 | BE+EN+3 | BY+1 | |

CONDITION: 4 ≠ 1

**a. STATEMENT OF THE RULE**




**b. EFFECT OF THE RULE ON TREES**

Figure 15: A Sample Transformational Rule: The Passive Transformation

The rule says that if you can analyze an intermediate phrase structure tree into a sequence consisting of a noun phrase, optionally an auxiliary verb, followed by a main verb and an object noun phrase, then you can transform the tree by moving the subject noun phrase (1) to the position of the object noun phrase (4) appending the word "by" on its left, moving the object noun phrase to subject position, and appending the morphemes, "be" and "en", to the left of the main verb. This rule changes the tree structure

corresponding to "Mary shot John" into that corresponding to "John was shot by Mary". (A later rule will move the "en" to the right of the next verb and a "post cyclic" rule will combine the two into a past participle.) The generation of a sentence by a transformational grammar consists of the generation of a deep structure tree by means of the context free base grammar and then transforming this tree through a series of intermediate structures into the surface structure tree by means of the transformational rules, which are usually ordered, marked as optional or obligatory, and applied cyclically to successive embedded clauses in complex sentences.

The transformational grammar appears capable of capturing the major syntactic facts about natural language, and a great deal of our current knowledge about the syntax of English has been discovered and codified in terms of this model. However, it is incredibly inefficient to parse with such a grammar and no parsing algorithm suitable for parsing any significant amount of text has ever been developed for this grammar model, although Stanley Petrick has spent considerable effort in this direction for a number of years and has probably the only working parsing algorithm for transformational grammars in existence (Petrick, 1965).

## Augmented Transition Networks

In order to obtain a grammar formalism with the linguistic adequacy of a transformational grammar while preserving the efficiency of the various context free parsing algorithms, I have been developing and refining a model of grammar which I call an augmented transition network (ATN). Presentations of this model appear in Woods (1969, 1970, 1973a). Earlier attempts along similar lines were made by Thorne, Bratley, and Dewar (1968) and by Bobrow and Fraser (1969). An ATN consists of a basic transition network grammar augmented with a set of registers which are carried along with the state and which can hold arbitrary pieces of tree structure, and with arbitrary conditions and actions associated with the arcs of the grammar which can test and set the contents of these registers. As a parsing proceeds with an ATN grammar, the conditions and actions associated with the transitions can put pieces of the input string into registers, use the contents of registers to build larger structures, check whether two registers are equal, etc. It turns out that this model can construct the same kinds of structural descriptions as those of a transformational grammar and can do it in a much more economical way. The merging of common parts of alternative structures, which the network grammar provides, permits a very compact representation of quite large grammars, and

this model has served as the basis for several natural language understanding systems such as the LUNAR system (Woods, Kaplan, and Nash-webber, 1972, Woods, 1973b). For speech understanding, the transition network grammar is one of the few linguistically adequate grammars for natural English that are at all amenable to coping with the combinatorial problems. This model is being used as the basis of the syntactic component of the BBN speech understanding project (Bates, 1974, Woods, 1974). Other types of context free grammars can be augmented by conditions and actions associated with the grammar rules in a similar way, but such grammars lose the benefits of the transition networks (such as merging common parts of different rules) which we discussed previously. Another advantage of the transition network formalism is the ease with which one can follow the arcs backwards and forwards in order to predict the types of constituents or words which could occur to the right or left of a given word or phrase. One of the important roles of a syntactic component in speech understanding is to predict those places where small function words such as "a", "an", "of" should occur since such words are almost always unstressed and difficult to unambiguously find in the input. In the BBN speech system such words are almost always found as a result of syntactic prediction and are not even looked for during lexical analysis since spurious matches would be found more often than correct ones.

The ATN formalism suggests a way of viewing a grammar as a map with various landmarks and recognizable locations that one encounters in the course of crossing a sentence from left to right. For speech understanding this perspective is beneficial, for example, in attempting to correlate various prosodic characteristics of sentences with such "geographical landmarks" within the structure of a sentence.

Let me conclude this presentation of syntactic techniques with a reiteration that I have not attempted to make a case that any one parsing technique or grammar formalism is uniformly better than others (indeed I do not believe there is a best one for all applications). Rather, I have attempted to give sufficient insight into the relative advantages and disadvantages to enable the reader to make appropriate choices for particular applications.

## Part II. Semantics

Turning now to the subject of semantics, I should perhaps first make the point that the word "semantics" means different things to different people. There is a tradition in philosophy and logic that specifies the semantics of formal systems such as the propositional calculus in terms of a set of "truth conditions" for each possible expression in the system. These truth conditions are abstract entities which specify the situations or "possible worlds" in which the statement would be true. In linguistics, on the other hand, concern is usually devoted to finding a notation or representation in which to specify ea of the different possible interpretations or "readings" which a natural language sentence can have and to procedures for determining whether a sentence is meaningful or "anomalous" (i.e. not meaningful). The linguist does not usually follow this up by providing a semantics in terms of truth conditions for his notation. In the field of programming languages in computer science, the semantics of a programming language is specified in terms of the computations which the machine is to perform as a result of a given expression. In specifying a formal semantics for such systems however, one usually takes recourse to defining the semantics by reducing it to another notation such as those of elementary arithmetic, whose semantics is presumably understood. In the fields of computational linguistics and artificial intelligence, the term is perhaps most misused. In some cases, it is taken to cover everything that isn't syntax -- i.e. everything that is not part of a grammar, while in others it is asserted to be no different in principle from syntax, and any basis for a distinction between the two is denied.

While I don't have the space here to go into a complete exposition of the different concerns of all of these different perspectives on semantics, I will try to give a brief synopsis of the distinctions.

Let us begin by considering what all of these different things which call themselves semantics have in common. According to m dictionary, semantics is "the scientific study of the relations between signs or symbols and what they denote or mean." This is the traditional use of the term and represents the common thread which links the different concerns discussed above. Notice that the term does not refer to the things den--t-d or the meanings, but to the <u>relations</u> between these t .ngs and the linguistic expressions which denote them. Thus, although it may be difficult to isolate exactly what part of a system is semantics, any system which understands sentences and carries out appropriate actions in response to them is somehow completing this connection, and therefore is applying semantic knowledge to this task. One of the common

misuses of the term semantics in the fields of computational linguistics and artificial intelligence is to extend the coverage of the term not only to this relation between linguistic form and meaning, but to all of the retrieval and inference capabilities of the system. This misuse arises since for many tasks in language processing, the use of semantic information to make an evaluation necessarily involves not only the determination of the object denoted, but also some inference about that object. In absence of a good name for this further inference process, terms such as "semantic inferences" have come to be used for the entire process. I regret to say that I have no really good substitute term for such processes and, since the terminology is so well established in some of the literature, I will use the term "semantic inferences" in this paper in referring to inferences that cross the boundary between symbol and referent and then draw conclusions about that referent. (One must be aware however that not all writers who use this term mean the same thing by it.)

The concerns of the linguists and the philosophers in the areas of semantics are effectively two halves of the same process, both of which the fields of computational linguistics and speech understanding will have to cope with. In reducing the semantics of natural language sentences to some formal notation, the linguist has only completed half of the job if he does not go on and specify a semantics of the resulting formal system. It is at this point that the concerns of philosophers and logicians in specifying the semantics for formal systems takes over. Notice that specifications of the formal semantics of programming languages in terms of the notations of elementary arithmetic are satisfactory only to the extent that we understand fully what these notations themselves mean. This is also the case for specifying the semantics of natural language.

I hope the above presentation has alerted you to some of the different kinds of things to which the term semantics can refer, and I will attempt to make clear which one I am using in the remainder of this presentation. I should point out that in the field of computational linguistics we don't have nearly as good an understanding of semantics as we do of syntax. I cannot give you the same kind of evolution of ideas through successively more powerful models and techniques, all of which are well understood. Here instead, the mechanisms which we understand thoroughly are known to be inadequate for dealing with many aspects of the problem, and the techniques which hold promise of dealing with some of the more difficult problems are not yet sufficiently understood or tested, for anyone to say whether they in fact solve the problem or not. In this area, then, we have many promising approaches, but few definite answers.

What I will attempt to do here is provide an understanding of some basic principles of semantic representation and interpretation that will apply to any system that understands natural language (whether text or speech), and then some specific techniques which I think have direct relevance to speech understanding. In particular, I will describe two techniques which are being applied in the BBN speech understanding system. One is the technique of semantic interpretation into procedural semantics which I have applied effectively to several natural language question-answering applications, and the other is the technique of "semantic intersections" in semantic network representations of knowledge which was developed by Quillian (1968, 1969). For more details on the specific applications of the latter technique to speech understanding, see Nash-Webber (1974 and 1975*). For the most part, the details of many other interesting things that are being done in the area of computational semantics for natural language will have to be left to the references. Articles which may be of interest include: Bruce (1973), Carbonell and Collins (1974), Collins and Quillian (1969), Collins and Warnock (1974), Fillmore (1968), Green and Raphael (1968), Heidorn (1972), Norman and Rumelhart (1973), Sandewall (1971), Winograd (1972), Woods (1967), and articles by Newell, Simmons, Wilks, Winograd, Schank, Colby, Abelson, Hunt, Lindsay, and Becker in Schank and Colby (1973).

## Procedural Semantics

It appears that the programming language theorists stand on firmer ground than the philosophers or the linguists in specifying the semantics of their systems, since they can define the semantics of their notations in terms of the procedures that the machine is to carry out. Notice that the notion of procedure shares with the notion of meaning that elusive quality of being impossible to present except by means of alternative representations. The procedure itself is something abstract which is instantiated whenever someone carries out the procedure, but otherwise, all one has when it is not being executed is some representation of it.

Although in ordinary natural language not every sentence is overtly dealing with procedures to be executed, it is possible nevertheless to use the notion of procedures as a means of specifying the truth conditions of declarative statements as well as the intended meaning of questions and commands. One thus picks up the semantic chain from the philosophers at the level of truth conditions and completes it to the level of formal specifications of procedures.

These can in turn be characterized by their operations on real machines and can be thereby anchored to physics. This notion of characterizing the truth conditions of sentences in terms of mechanical procedures is one that I called "procedural semantics" in my 1968 AFIPS paper (Woods, 1968) and the term has since gained wide circulation. The application of this technique in computer systems for natural language understanding has been very effective. Two notable computer systems which make use of this type of semantics are the LUNAR system (Woods, Kaplan, & Nash-Webber, 1972, Woods, 1973b) and the blocks world system of Winograd (1972). The former understands and answers questions such as "What is the average concentration of aluminum in high alkali rocks?", while the latter understands and carries out instructions such as "Put the pyramid on the block in the corner," (including resolving the ambiguity by determining whether there is a pyramid on a block or a block in the corner). Since the semantic techniques used in the LUNAR system are more formalized and rule driven and since I am more familiar with the details of that system, I will use LUNAR as the principal illustration of the technique. I think the rules used there can effectively serve as a formal model for what is going on in a number of other language understanding systems.

## Semantics in LUNAR

The semantic framework of the LUNAR system consists of three parts -- a semantic notation in which to represent the meanings of the sentences, a specification of the semantics or meanings of this notation by means of LISP programs, and a procedure for assigning representations in the notation to input sentences. In LUNAR, the semantic notation (which I have referred to there as a query language) consists of an extended notational variant of the predicate calculus.

The query language contains essentially three kinds of constructions:

1) designators, which name or denote objects or classes of objects in the data base,

2) propositions, which correspond to statements that can be either true or false in the data base, and

3) commands, which initiate and carry out actions.

Designators come in two varieties -- individual specifiers and class specifiers. Individual specifiers correspond to proper nouns and variables. For example, S10046 is a designator for a particular sample, OLIV is a designator for

a certain mineral (olivine), and X3 can be a variable denoting any type of object in the data base. Class specifiers are designators used to denote classes of individuals over which quantification can range. They consist of the name of an enumeration function for the class plus arguments. For example, (SEQ TYPECS) is a specification of the class of type C rocks (i.e. breccias) and (DATALINE S10046 OVERALL OLIV) is a specification of the set of lines of a table of chemical analyses which correspond to analyses of sample S10046 for the overall concentration of olivine.

Elementary propositions are formed from predicates with designators as arguments, and complex propositions are formed from these by use of the logical connectives AND, OR, and NOT and by quantification. For example, (CONTAIN S10046 OLIV) is a proposition formed by substituting designators as arguments to the predicate CONTAIN, and (AND (CONTAIN X3 OLIV) (NOT (CONTAIN X3 PLAG))) is a complex proposition corresponding to the assertion that X3 contains olivine but does not contain plagioclase. Elementary commands consist of the name of a command function plus arguments, and like propositions, complex commands can be constructed using logical connectives and quantification. TEST is a command function for testing the truth value of a proposition given as its argument. Thus, (TEST (CONTAIN S10046 OLIV)) will answer yes or no depending on whether sample S10046 contains olivine. Similarly PRINTOUT is a command function which prints out a representation for a designator given as its argument.

The format for a quantified proposition or command is:

(FOR QUANT X / CLASS : PX ; QX )

where QUANT is a type of quantifier (EACH, EVERY, SOME, THE, numerical quantifiers, etc.), X is a variable of quantification, CLASS is a class specifier for the class of objects over which quantification is to range, PX specifies a restriction on the range, and QX is the proposition or command being quantified. (Both PX and QX may themselves be quantified expressions.) For example (FOR EVERY X1 / (SEQ TYPECS) : (CONTAIN X1 PLAG) ; (CONTAIN X1 OLIV)) is a quantified proposition corresponding to the statement that every type C rock that contains plagioclase also contains olivine. (FOR EVERY X2 / (DATALINE S10046 OVERALL OLIV) : T ; (PRINTOUT X2)) is a quantified command to printout all of the chemical analyses of S10046 for overall olivine concentrations. (For expository reasons, the notation has been slightly simplified here compared to that actually used in the LUNAR system, but the differences are minor.)

Semantics of the Notation

Having specified our semantic notation for representing the meanings, we must now specify the meanings of our notations. As mentioned before, we do this in LUNAR by relating the notations to procedures which can be executed. For each of the predicate names that can be used in specifying semantic representations, we will specify a procedure or subroutine which will determine the truth of the predicate for given values for the arguments. Similarly for each of the functions which can be used, we will specify a procedure which can compute the value of that function given the values of its arguments. For each of the class specifiers for the FOR function, we will require a subroutine which enumerates the members of the class. The FOR function itself is also defined by a subroutine as are the logical operators AND, OR and NOT and the basic command functions TEST and PRINTOUT. Thus any well formed expression in the query language is a composition of functions which have procedural definitions in the retrieval component and are therefore themselves well defined procedures capable of execution on the data base. In fact in the LUNAR system, the definition of all of these procedures is done in LISP and the notation of the query language is so chosen that its expressions are executable LISP programs. The totality of these function definitions and the data base on which they operate constitute the retrieval component of the system.

It should be pointed out that by virtue of this definition of the primitive functions and predicates as LISP functions, the query language can be viewed simultaneously as a higher-level programming language and as an extension of the predicate calculus. This gives rise to two different possible types of inference for answering questions, corresponding to the philosopher's distinction between intension and extension. First, because of its definition by means of procedures, a question such as "Does every sample contain silicon?" can be answered <u>extensionally</u> (that is by appeal to the individuals denoted by the class name "samples") by enumerating the individual samples and checking whether sodium has been found in each one. On the other hand, this same question could have been answered <u>intentionally</u> (that is by reference to its meanings alone without reference to the objects denoted) by means of the application of inference rules to other (intentional) facts such as the assertion "Every sample contains some amount of each element." Thus the expressions in the query language are capable either of direct execution against the data base (extensional mode) or manipulation by mechanical inference algorithms or theorem provers (intentional mode). Only the former (extensional) mode of inference is actually used in the LUNAR system. This gives rise to some limitations (e.g.

it is not possible to prove most assertions about infinite sets in extensional mode), but is very efficient for a variety of question-answering applications.

## Semantic Interpretation

Having now specified the notation in which we will represent the meanings of English sentences in our system and making sure that we understand the nature of the meanings of the expressions in that notation, we are now left with the specification of the process whereby meanings are assigned to sentences. This process is referred to as semantic interpretation, and in LUNAR it is driven by a set of formal semantic interpretation rules. The semantic interpreter operates on a syntactic structure or fragment of one which has been constructed by the parser, assigning semantic expressions in the notation to the nodes of this structure to indicate the "meanings" of those constructions to the system. In LUNAR this procedure is such that the interpretation of nodes can be initiated in any order, but if the interpretation of a node requires the interpretation of a constituent node, then the interpretation of that constituent node is performed before the interpretation of the higher node is completed. Thus, it is possible to perform the entire semantic interpretation by calling for the interpretation of the top node (the sentence as a whole), and this is the normal mode in which the interpreter is operated in the LUNAR system.

## Semantic Rules

In determining the meaning of a construction, two types of information are used -- syntactic information about sentence construction and semantic information about constituents. For example, in interpreting the meaning of the sentence, "S10046 contains silicon," it is both the syntactic structure of the sentence (subject = S10046; verb = "contain"; object = silicon) plus the semantic facts that S10046 is a sample and silicon is a chemical element that determine the interpretation (CONTAIN S10046 SILICON). (Note that the predicate CONTAIN here is the name of a procedure in the retrieval component and it is only by the "accident" of mnemonic design that its name happens to be the same as the English word "contain" in the sentence that we have interpreted.)

In LUNAR, this information about the semantic interpretations of syntactic structures is embodied in semantic rules consisting of patterns that determine whether

a rule can apply and actions that specify how the semantic interpretation is to be constructed. An example of such a rule is given in Figure 16.

```
(S:SAMPLE-CONTAIN
        (S.NP (MEM 1 (SAMPLE)))
        (S.V (OR (EQU 1 HAVE)
                 (EQU 1 CONTAIN)))
        (S.OBJ (MEM 1 (ELEMENT OXIDE ISOTOPE)))
        →
        (PRED (CONTAIN (# 1 1)(# 3 1))))
```

Figure 16: A Sample Semantic Interpretation Rule

The name of the rule is S:SAMPLE-CONTAIN, and the left-hand side, or pattern part of the rule, consists of three templates which match fragments of syntactic structure. The first template requires that the sentence being interpreted have a subject noun phrase which is a member of the semantic class SAMPLE, the second requires that the verb be either "have" or "contain", and the third requires a direct object which is either a chemical element, an oxide or an isotope. The terms S.NP, S.V and S.OBJ name schemata for tree fragments which are used not only to test for the presence of their corresponding syntactic structures in the sentence, but also to associate reference numbers with selected nodes in the structure. These numbers are used for reference by the semantic conditions in the templates and for use in the right-hand side of the semantic rule. For example, the tree fragment S.NP locates the subject noun phrase of the sentence and associates the reference number 1 with that noun phrase.

The right-hand side, or action part, of the rule follows the right arrow and specifies that the interpretation of this node is to be a predicate formed by inserting the interpretations of two constituent nodes into the schema (CONTAIN (# 1 1)(# 3 1)), where the expressions (# m n) refer to the interpretation of the node with reference number n for template number m in the match of the left-hand side of the rule.

### Organization of Rules

The semantic rules for interpreting sentences are usually governed by the verb of the sentence. That is, out of the entire set of semantic rules, only a relatively small number of them can possibly apply to a given sentence because of the verb mentioned in the rule. Similarly the rules which interpret noun phrases are governed by the head noun of the noun phrase. For this reason, the semantic rules in LUNAR are indexed according to the heads of the constructions to which they could apply and recorded in the dictionary entry for the head words. Each rule then characterizes a syntactic/semantic environment in which a word can occur and specifies its interpretation in that environment. The templates of a verb rule thus describe the necessary and sufficient constituents and semantic restrictions in order for the verb to be meaningful. Nouns in noun phrases behave similarly. That is, the semantic rules not only specify the process of interpretation which assigns semantic representations, but their left-hand sides also specify the conditions under which given words and constructions are meaningful.

### Semantic Rules in General

The above presentation is oversimplified in a number of respects for the sake of expository brevity. There is in general a greater variety of devices that are used in the semantic rules of the LUNAR system, and there are numerous details of operation that we will not consider here. (Such as the desired behavior when a template or a rule matches in several different ways.) For more details on these issues, the reader is referred to Woods (1967) and to Woods, Kaplan, and Nash-Webber (1972). There are also many other interesting issues in the semantics of natural language which have not been explored in the LUNAR system or any other computer system which are currently more the domain of philosophers than computer scientists but which will eventually have to be handled by computer systems if they are to be facile at understanding human language. The diversity of these issues, however, is beyond the scope of this presentation.

In many question answering systems semantic interpretation rules are paired more directly with the syntactic rules of the grammar so that there is little or no template matching required (and consequently less latitude for producing semantic interpretations that are not in node-for-node correspondence with the syntactic structure). In still other systems, the semantics are not formalized in rules, but are simply embodied in arbitrary computer programs (and consequently totally unconstrained in what

could be done theoretically but providing little or no theory or conceptual framework for what is going on.) However, the kind of semantic rules that are used in LUNAR can be used as formal models to explain what is going on in the semantics of these other systems in which the semantics is either more restricted or less formalized.

## Semantic Judgments

As in the case of syntax, semantics has both a judgmental and a structural aspect. That is, semantic information is used both to construct semantic representations of the meanings of the sentences and to reject anomalous or semantically ill-formed sentences. What we have described so far has mostly dealt with the structural aspect -- how to assign a semantic representation to a sentence and what representation to assign. This capability is necessary for any language understanding system whether it is text or speech. In the judgmental component, however, there are a number of things which semantics can do which are particularly important for speech understanding. As we pointed out above, the pattern parts of the semantic interpretation rules can be used to specify what assemblages of syntactic structures and lexical words are meaningful.

In the next few sections, what I would like to do is briefly survey the uses of semantic information which have been made in various question answering systems using the notion of semantic interpretation rules as presented above to unify the discussion. I shall no longer be directly concerned with the use of the rules for the assignment of semantic interpretations to sentences, but with the ancillary use of the information embodied in these rules for other purposes.

Semantic information is used in a number of text oriented language understanding systems to select semantically meaningful parsings from among all of the possible parsings of a sentence. For example, in the context of airline flight schedules in interpreting a sentence such as "Does American have a fight from some east coast city to Chicago" we can tell that the phrase "to Chicago" modifies flight and not city because we have semantic interpretation rules for flights to places while we do not have any rules to interpret cities to places. In speech understanding, this ability to determine whether a given interpretation of a sentence is semantically meaningful is critical not only for choosing between alternative parsings, but also for choosing between alternative segmentations of the input signal into words.

In the next few sections I will discuss some of the techniques that have been used in various question answering systems to use semantic information for this judgmental role and discuss their advantages and limitations for speech understanding applications.

## Semantic Selectional Restrictions

As we mentioned above, the attempt to characterize the difference between semantically well-formed sentences and those which are semantically anomalous has been a major concern of many linguistic semanticists (see e.g. Katz & Fodor, 1964). The device which is used in most such attempts is a notion of semantic selectional restrictions -- restrictions between the verbs of a sentence and semantic features of the arguments which they can sensibly take. For example, the restriction that verbs like "intend" require higher animate subjects is used to explain the oddness of sentences such as "the rock intends to sit there." This account assumes that the nouns of the language can be assigned to semantic classes such as "higher animate" and that there must be "semantic agreement" or at least no semantic disagreement between the verb of a sentence and the subjects, objects and other arguments which it can take. It is in this area of semantics that the misconceptions about the distinction between syntax and semantics arise, since there is usually no difference in principle between the implementation of such semantic restrictions to reject semantically anomalous sentences and implementation of syntactic restrictions such as number agreement to reject syntactically incorrect sentences. For sufficiently restricted and fixed domains of discourse, it is possible to implement such semantic selectional restrictions by subcategorizing the syntactic categories of the grammar with classes like ´animate noun´ and ´color adjective´ rather than simply noun or adjective. One thereby incorporates the testing of semantic selectional restrictions into the grammar and avoids the need for any special mechanism for testing semantic selectional restrictions.

The technique of semantically subcategorizing the syntactic categories of the grammar has been applied effectively in limited speech understanding applications. It has the advantage of being efficient in execution and easy to implement for sufficiently simple understanding tasks. However, one should understand its limitations. One of the major inadequacies is that the use of semantic selectional restrictions as prerequisites for grammaticality or semantic well-formedness is not quite correct. Rather most such conditions are required only for a sentence to be true. When the sentence is a question or when it asserts a

negative possibility, then semantic selectional restrictions may be violated by perfectly reasonable sentences. A speech understanding system which contains such restrictions embedded in its grammar will fail to parse such inputs. (For example, in Terry Winograd's blocks world program the sentence "Can a table like blocks?" fails to parse since the system applies the selectional restriction that "like" requires an animate subject.) A speech understanding system which used such selectional restrictions as a prerequisite for acceptability of an interpretation of a speech signal would be unable to "hear" this sequence of words no matter how well articulated and how successful the acoustic and phonological analysis, but would rather insist on looking for some other interpretation of the signal.

An additional limitation of the semantic selectional restriction approach is that the necessary semantic information associated with a given argument to a verb is not necessarily associated with the lexical items in the noun phrase, but may be associated with the referent of the noun phrase instead. The association of such information with the dictionary entries for the words is really just an approximation (albeit a useful one for many applications) of what one really wants the semantic selectional restrictions to test.

A major practical difficulty with incorporating the semantic selectional restrictions into the syntactic categories of the grammar is the lack of extendability thus induced. If one wants to apply the system to a different domain of discourse or to extend the domain slightly, he has to redefine the categories of the grammar.

## Semantic Screening

A somewhat more versatile technique for using semantic information to select an appropriate parsing is to apply semantic rules to the nodes of the syntactic tree structure as the nodes are built by the parser. If the node just constructed fails to have a semantic interpretation, then that computation path of the parser is rejected and the parser looks for other ways to parse the input. This technique of semantic screening applies the semantic selectional restrictions as a filter or a sieve during the parsing operation. In its simplest form, the semantic rules are associated with the rules of a context-free grammar in a one-to-one fashion so that as soon as a syntactic rule is applied, the corresponding semantic rule is tested. Semantic screening is often touted as a mechanism for gaining increased efficiency in parsing since it tends to cause early rejection of parsing paths which otherwise would

have been continued further. This argument, however, neglects to count the cost of the semantic interpretation on uncompleted parsings which would not have been completed in any case for syntactic reasons. Whether semantic screening really provides an increase in efficiency depends on the relative costs of the extra or unnecessary semantic processing and the syntactic processing that is thereby eliminated. In many situations, it is more efficient to complete the syntactic analysis and then apply the semantic testing.

Another technique which is related to semantic screening is to apply tests not only of general semantic well-formedness but also tests of factuality in conjunction with the formation of a constituent. This is the case for example in Winograd's system when he makes his decision about "put the pyramid on the block in the corner" on the basis of whether there is a pyramid on a block in the current state of the world and not just on the basis of general information about whether pyramids can be on blocks. This technique can be very useful in some situations, but its exclusive and uncontrolled use would make it impossible to say things that were not already true or to ask about things that were not true.

## Semantic Selection

A major inadequacy of semantic (and of factual) screening and indeed of any application of semantic selectional restrictions as strict prerequisites for well-formedness is its inability to deal with sentences such as "I saw the man in the park with a telescope" in which there are many possible parsings which are all semantically possible, but are not equally plausible. Although it is possible that I was in a park which contained a telescope when I saw the man somewhere else, this is not the most likely interpretation in absence of specific information that would indicate this interpretation. Rather there is a kind of default interpretation that the telescope was probably used to see the man, and in absence of reason to believe otherwise the man was probably in the park. What is required in general rather than a mere rejection of semantically ill-formed interpretations is a mechanism to select the most plausible interpretation from among a set of syntactically related alternatives. Although the solution of this problem in general is not at hand, a beginning has been made in a mechanism called selective modifier placement in the LUNAR parser (see Woods, 1973a). This mechanism uses information such as the fact that a telescope is an optical instrument and one can see with an optical instrument to prefer the alternative of "with a telescope" modifying

"see", while in absence of semantic preference, the modifier "in the park" modifies the syntactically preceding noun phrase "man". The technique has not been systematically developed, however, and except for the placement of prepositional phrase modifiers, the use of semantic judgments in LUNAR to select among alternative parsings is not well developed.

## Semantic Prediction

All of the preceding techniques for making semantic judgments about completed syntactic constructions are of great importance for speech understanding. There are, however, situations in the course of understanding a speech utterance where one does not have a complete construction to work with and would like to make use of semantic information to guide the speech understander to look for words which might have been slightly garbled or to provide initial preferences among the words that are discovered on the basis of acoustic and lexical analyses alone. Given for example that we have found the words "sample" and "contain" in a speech signal, we would like to make use of our semantic information to predict that there should now occur a word which is a chemical element, an oxide or an isotope. This information is contained in our semantic rules (specifically it is in the left-hand sides of the rules). Similarly upon encountering the words "sample" and "contain" among a large number of other words in the initial word lattice, we would like to use the semantic information to notice that these two words are related and perhaps go together in the interpretation of the utterance. Both of these semantic roles make use, not of the logical or interpretative sense of semantics, but of a kind of associational semantics which studies the semantic relationships among words and concepts. There are a number of psychologists and psycholinguists as well as people in artificial intelligence, sociology and other fields who have been trying to model this aspect of semantics with various kinds of network structures. The initial impetus in this area was created by Ross Quillian (1968, 1969), but other researchers in this area of semantics include Abelson, Carbonell, Collins, Rumelhart and Norman, Schank, Simmons, and others (a sampling of most of these authors is given in Schank & Colby, 1973 and others are cited explicitly in this paper.) The work of Fillmore (1968) has also been influential in this area of study, and recently, similar notions have been used at MIT as the basis for programs that analyze visual scenes (Winston, 1970). I will describe here some of the characteristics of semantic networks as Quillian visualized them which have direct application in speech understanding and which have been included in the BBN speech understanding system.

Quillian was not interested in the notions of semantics as characterizing truth. Indeed he denied (I think erroneously) the psychological relevance of such notions. Rather he viewed the "meaning" of a word as merely a collection of the concepts that are associated with it (without, however, giving any adequate explication of what was meant by a concept). I consider Quillian's original formulation and much of the work that it has stimulated to be inadequate in the respect that it doesn't give any attention to a specification of the semantics of the network notation itself, but that doesn't lessen the validity of many of the points that he and others of this school have raised.

Quillian was concerned with investigating the structure in which humans store information in their brains. Thus, the so called semantic networks are really attempts at finding structures and organizations for storing knowledge. His concern is not with having a notation in which to write down a list of facts, but rather with an overall memory structure in which the interrelationships among those facts, which humans use for retrieval of information and for construction of inferences, are explicitly and efficiently represented. The important thing for Quillian is not so much the structure of a particular concept, but the network of relations to other concepts that are established. In particular, Quillian sought to devise a mechanism and a structure which would account for the types of semantic associations which people make and the way these associations manifest themselves in human language understanding.

To give a flavor of the kind of network that Quillian had in mind, Figure 17 (taken from Quillian, (1969)), gives an example of the concept associated with the lexical item "client". Each lexical item or word points to one or more "concepts" or nodes in the semantic net (corresponding to different senses of the word) each of which is merely an assemblage of pointers to other concept nodes in the network. In Figure 17 the identifiers PERSON, EMPLOY, and PROFESSIONAL stand for pointers to other concept nodes in the network. In Quillian's view, the meaning of a concept is the sum total of the collection of concept nodes to which it is connected -- no more and no less. While this gives very little leverage on solving any of the problems of semantic interpretation or characterization of truth conditions, it is a superb mechanism for accomplishing the semantic predictions and noticing the coincidences among semantically related words that are required for speech understanding. In particular, Quillian's notion of semantic intersection can play an important role in speech understanding.

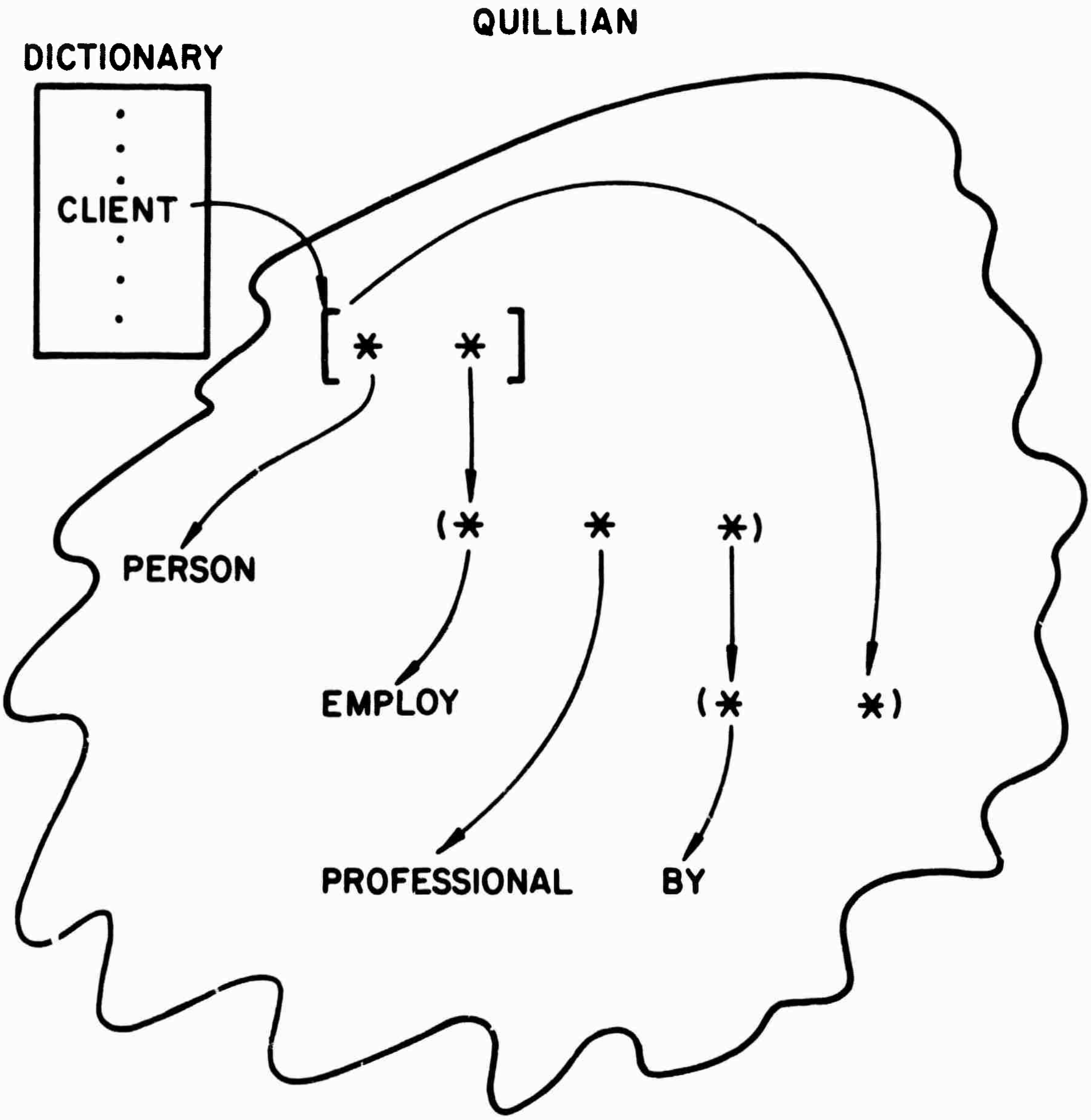



Figure 17: A Fragment of a Quillian Network

## Semantic Intersection

Quillian developed the notion of semantic intersection as an attempt to account for the human capability to immediately identify the relationships between diverse things such as between 'plant' and 'alive' or (more subtly) between Madrid and Mexico, and to account for the tendency of people to accept an ambiguous term in a particular sense induced by the appropriateness to the context without noticing the other possible senses (a phenomenon called "foregrounding"). In foregrounding, the appropriate sense is somehow brought forward and made more accessible than the other senses due to the influence of the context. The mechanism which Quillian proposed to account for such phenomena and which he believed was the principal process for accessing information from one's knowledge store was a process which he called semantic intersection. Quillian assumed that in the brain, whenever a concept was brought into consideration in a discourse or whatever, it was somehow stimulated or "activated" and that this activation passed out in waves from the source of the stimulation to the concept nodes to which it was connected. When the activation waves from two different sources met at some node in the memory, a semantic intersection was detected, and a path through the semantic memory was thereby established which represented the semantic relationship between the two source concepts. (e.g. Madrid is in Spain which is like Mexico in language and culture.) Similarly, such activations have some duration in time, and when an ambiguous word is encountered, the sense that people are likely to take is the sense which has semantic connections with concepts that are currently activated (as detected by the presence of semantic intersections).

In speech understanding, this foregrounding effect of semantic intersections can be used to influence the words that one hears in an otherwise ambiguous segment of speech, and can be used to detect the coincidences of semantically related words in a word lattice. Following connections through the semantic network can also be used to predict words that have not been detected in the signal but which are sufficiently likely that they should be looked for. For details on the use of such techniques in the understanding of continuous speech, the reader is again referred to Nash-Webber (1974, 1975*). Notice that the information that we have in the pattern parts of the semantic interpretation rules of LUNAR is one type of information that we would like to have in such a semantic network. Notice also that whereas in LUNAR the information about associated semantic classes is available conveniently if one starts with the head of a construction, similar information in a semantic network format would be equally accessible from any of the concepts involved in the rule. This is one more instance of

the importance of breaking a priori orderings of processing in speech understanding in favor of multiple, redundant ways of achieving the same result. In any given utterance, it could be one of the critical head words that is garbled, and one would like to be able nevertheless to find the semantic relationships among the arguments and use them to predict the missing head.

## Other Aspects of Semantic Nets and Knowledge Representation

Another notion embedded in Quillian's conception of a semantic network (which also has rudimentary beginnings in Raphael's SIR system (Raphael, 1964)) is that information about a concept can be stored at several different levels up a chain of more and more inclusive concepts (Quillian called them superconcepts). For example, a canary is a bird which is a type of animal which is in turn a physical object. It may thus have certain properties which are stored directly at the level of canary (such as being yellow) but other properties that are common to a great many concepts and which are stored at the most general level of applicability. These would be automatically inherited by subconcepts (in absence of contrary information) without having to be stored over and over again for each of the entities for which they are true.

There is a tremendous amount of interest right now in various semantic network representations, what such structures should look like, how they should be used to do inferences, what kinds of things should be put into a network in response to understanding a sentence, etc. In particular, it is pointed out, notably by Schank and his students, that a great deal of what is understood in response to an input sentence comes from gratuitous assumptions that are made on the basis of knowledge already in memory and not specifically transmitted by the sentence. For example, Schank cites dialog pairs such as "would you like an ice cream cone?" and "I just ate", in which the second utterance should be interpreted as giving a negative answer to the question. I think it should be apparent that when one attempts to understand spoken discourses and make judgments about the contextual appropriateness of a given interpretation of an utterance, the ability to make such semantic inferences using large amounts of semantic and factual knowledge (as well as pragmatic knowledge about what the speaker is likely to say in a given situation) will be of paramount importance. The inability to account for a given interpretation of an utterance by being able to relate it to what has been said before or to some aspect of the current context should raise the possibility that the

utterance has been misheard. The ability to fully use this level of sophisticated inference as part of a speech understanding system, however, will probably have to await further developments in the ongoing studies in knowledge representation and mechanical inference. The techniques which exist today in these areas are either extremely limited or inordinately cumbersome.

## CONCLUSION

I have attempted here to provide a perspective on some of the work that has been done in the areas of syntax and semantics for understanding natural language by machines and to call special attention to those techniques which have particular relevance to the problems of speech understanding.

I have tried to cover a range of different parsing algorithms and grammar models with emphasis on the advantages and disadvantages of the various features of these models for the particular types of problems that one will encounter in analyzing continuous speech, and I have tried to give my opinions as to the value of these different features. In particular, I have argued that the use of a predetermined order of finding things (such as left to right across the sentence) is potentially dangerous and perhaps to be avoided. I have pointed out that the ambiguity of syntactic word class, which is one of the major sources of ambiguity in English text) is greatly magnified in speech understanding by the inability to uniquely determine what the word at a given position is. Whereas in text parsing, one at least knows what the word is and therefore has an expectation of two or three possible syntactic categories for it, in speech understanding we may have a half dozen alternative possible words at a given point, each with one or more possible syntactic categories. Hence the combinatorial problems that arise from the multiplication of possible alternative analyses is much worse for speech. This is complicated by the fact that most of the techniques that have been developed in text parsers for minimizing the impact of these combinatorial possibilities require carefully designed sequences of looking for things which conflict with the above observation that such constrained orderings are sensitive to the errors in the lexical analysis of the input that are virtually inevitable in speech.

The use of word lattices as input instead of sequences and the design of parsing algorithms around well-formed substring tables or charts appear to be viable methods for dealing with the combinatorial problem of speech understanding. The merging of common parts of different analyses permitted by transition network grammars is also helpful in this respect. In order to be able to correct errors, it will be essential to be able to come at a given parsing from several directions. Consequently checks will be necessary at appropriate points to avoid duplicating an analysis that has already been found.

Another important role of syntax in a speech understanding system is the prediction of those places where

small function words might occur in order to compensate for the unreliability of their identification by lexical analysis.

Although our understanding of semantics is not as well advanced as that of syntax (which itself is far from complete), there are a number of semantic techniques that language understanding programs have used which can have great benefit in the construction of speech understanding machines. These include the use of procedural semantics for the specification of the operations which are to be carried out in response to the understanding of the sentence, the use of semantic selectional restrictions to rule out unlikely interpretations of the speech signal, and the use of semantic associations as embodied in the Quillian semantic intersection technique to notice coincidences between semantically related words at different points in the input. One should be aware, however, of the limitations of some of these techniques and the need for continued research in the areas both of syntax and semantics in order to increase the range of things which such systems can understand and their abilities to choose correctly between alternative interpretations of a signal.

I think it is clear that in order to cover the scope of material that I have it has been necessary to treat many issues rather shallowly and others not at all. Hopefully the references will provide additional detail for the interested reader to follow up. I hope that I have given you some feeling for the issues and some of the things that are going on in computational linguistics, linguistics, psychology, and artificial intelligence relative to syntax and semantics and some of the ramifications of the speech understanding task for these areas. Given the different perspective that the speech understanding task places on the roles of syntax and semantics in language understanding, I believe that the speech understanding problem can have almost as great an impact on research in syntax and semantics as these areas are now having on the problem of automatic speech recognition.

## References

[1] Bates, M. (1974)
"The Use of Syntax in a Speech Understanding System," *Proceedings of IEEE Symposium on Speech Recognition*, Carnegie-Mellon University, Pittsburgh, Penn. pp. 226-233.

[2] Bobrow, D.G. and Fraser, J.B. (1969)
"An Augmented State Transition Network Analysis Procedure," *Proceedings International Joint Conference on Artificial Intelligence*, Washington D.C., pp. 557-567.

[3] Bruce, B. (1973)
"Case Structure Systems," *Proceedings of Third International Joint Conference on Artificial Intelligence*, Stanford University, Stanford, California, pp. 364-371.

[4] Carbonell, J.R. and Collins, A.M. (1973)
"Natural Semantics in Artificial Intelligence," *Proceedings of Third International Joint Conference on Artificial Intelligence*, Stanford University, Stanford, California, pp. 344-351.

[5] Chomsky, N. (1965)
*Aspects of the Theory of Syntax*, MIT Press, Cambridge, Mass.

[6] Collins, A.M. and Quillian, M.R. (1969)
"Retrieval Time from Semantic Memory," Journal of Verbal Learning and Verbal Behavior, 8 (2), pp. 240-247.

[7] Collins, A.M. and Warnock, E.H. (1974)
Semantic Networks, Report 2833, Bolt Beranek and Newman Inc., Cambridge, Mass.

[8] Denes, P.B. and Pinson, E.N. (1963)
*The Speech Chain*, Bell Telephone Laboratories, Inc.

[9] Earley, J. (1970)
"An Efficient Context-Free Parsing Algorithm," CACM 13 (2), pp. 94-102.

[10] Fillmore, C.J. (1968)
"The Case for Case," in Bach, E. and Harms, R. (eds.) *Universals in Linguistic Theory*, Holt, Rinehart and

winston, New York.

[11] Floyd, R.W. (1967)
"Nondeterministic Algorithms," JACM 14 (4) pp. 636-644.

[12] Green, C.C. and Raphael, B. (1968)
"The Use of Theorem-Proving Techniques in Question-Answering Systems," Proc. 1968 ACM National Conference, pp. 169-181.

[13] Greibach, S.A. (1967)
"A Simple Proof of the Standard-Form Theorem for Context-Free Grammars," in Mathematical Linguistics and Automatic Translation, Report NSF-18, Harvard University Computation Laboratory, Cambridge, Mass.

[14] Griffiths, T. and Petrick, S.R. (1965)
"On the Relative Efficiencies of Context-Free Grammar Recognizers," CACM 5 (8), pp. 289-300.

[15] Hays, D.G. (1962)
"Automatic Language-Data Processing," in Harold Borko (ed.), Computer Applications in the Behavioural Sciences, Prentice Hall, Englewood Cliffs, New Jersey.

[16] Heidorn, G.E. (1972)
Natural Language Inputs to a Simulation Programming System, Ph.D. Thesis, Yale University, New Haven, Conn.

[17] Jakobson, R., Fant, C.G., and Halle M. (1967)
Preliminaries to Speech Analysis, MIT Press, Cambridge, Mass.

[18] Katz, J.J. and Fodor, J.A. (1964)
"The Structure of a Semantic Theory," in Katz, J.J. and Fodor, J.A. (eds.) The Structure of Language: Readings in the Philosophy of Language, Prentice-Hall, Englewood Cliffs, New Jersey, pp. 479-518.

[19] Kay, M. (1967)
"Experiments with a Powerful Parser," Memorandum, RM-5452-PR, The RAND Corporation, Santa Monica, California.

[20] Kuno, S. and Oettinger, A.G. (1963)
"Multiple-Path Syntactic Analyzer," Information Processing 62, North-Holland,

Amsterdam, pp. 306-312.

[21] Nash-Webber, B. (1974)
"Semantic Support for a Speech Understanding System," Proceedings of IEEE Symposium on Speech Recognition, Carnegie-Mellon University, Pittsburgh, Penn., pp. 244-249.

[22] Nash-Webber, B. (1975*)
"The Role of Semantics in Automatic Speech Understanding", in Representation and Understanding, Bobrow, D.G. and Collins, A. (eds.) Academic Press. (in press)

[23] Newell, A. et al. (1973)
Speech Understanding Systems: Final Report of a Study Group, North-Holland/American Elsevier, Amsterdam.

[24] Norman, D.A. and Rumelhart, D.E. (1973)
"Active Semantic Networks as a Model of Human Memory," Proc. Third International Joint Conference on Artificial Intelligence, Stanford University, Stanford, California, pp. 450-463.

[25] Petrick, S.R. (1965)
A Recognition Procedure for Transformational Grammars, Ph.D. Thesis, Department of Modern Languages, M.I.T., Cambridge, Mass.

[26] Quillian, M.R. (1968)
"Semantic Memory," in Minsky, M.L. (ed.) Semantic Information Processing, MIT Press, Cambridge, Mass.

[27] Quillian, M.R. (1969)
"The Teachable Language Comprehender: A Simulation Program and Theory of Language," CACM 12 (8), pp. 459-476.

[28] Raphael, B. (1964)
"A Computer Program which 'Understands'," AFIPS Conference Proceedings, Vol. 26 (1964 FJCC) pp. 577-589.

[29] Sandewall, E. (1971)
"A Programming Tool for Management of a Predicate-Calculus-Oriented Data Base," Proc. Second International Joint Conference on Artificial Intelligence, The British Computer Society, London, pp. 159-166.

[30] Schank, R.C. and Colby, K.M. (1973)
Computer Models of Thought and Language, W.H. Freeman

& Co., San Francisco, California.

[31] Thorne, J., Bratley, P. and Dewar, H. (1968) "The Syntactic Analysis of English by Machine," in D. Michie (ed.) Machine Intelligence 3, American Elsevier, New York, N.Y.

[32] Winograd, T. (1972) Understanding Natural Language, Academic Press, New York, N.Y.

[33] Winston, P.H. (1970) "Learning Structural Descriptions from Examples," MAC TR-76, MIT Project MAC, Cambridge, Mass.

[34] Woods, W.A. (1968) "Procedural Semantics for a Question-Answering Machine", AFIPS Conference Proceedings, Vol. 33 (1968 FJCC), pp. 457-471.

[35] Woods, W.A. (1967) Semantics for a Question Answering System, Report NSF-19, The Computation Laboratory, Harvard University, Cambridge, Mass. (NTIS number PB-176-548).

[36] Woods, W.A. (1969) "Augmented Transition Networks for Natural Language Analysis," Report CS-1, Computation Laboratory, Harvard University, Cambridge, Mass.

[37] Woods, W.A. (1970) "Transition Network Grammars for Natural Language Analysis," CACM 13 (10), pp. 591-606.

[38] Woods, W.A. (1973a) "An Experimental Parsing System for Transition Network Grammars," in R. Rustin (ed.), Natural Language Processing, Algorithmics Press, New York, New York.

[39] Woods, W.A. (1973b) "Progress in Natural Language Understanding: An Application to Lunar Geology," AFIPS Conference Proceedings, Vol. 42, (1973 National Computer Conference) pp. 441-450.

[40] Woods, W.A. (1974) "Motivation and Overview of BBN SPEECHLIS: An Experimental Prototype for Speech Understanding Research", Proc. IEEE Symposium on Speech Recognition, Carnegie-Mellon University, Pittsburgh, Penn., pp. 1-10.

[41] Woods, W.A., Kaplan, R.M. and Nash-Webber, B. (1972) "The Lunar Sciences Natural Language Information System: Final Report", BBN Report No. 2378, Bolt Beranek and Newman Inc., Cambridge, Mass. (NTIS number N72-28984).

[42] Younger, D.H. (1966) "Context-Free Language Processing in Time n3," <u>Proceedings 1966 Annual Symposium on Switching and Automata Theory</u>, IEEE Conference Record 16 C 40, 1966, pp. 7-20.